CVact™
User Guide

CADDS® 5i Release 14

DOC40037-014

Parametric Technology Corporation

Copyright © 2005 Parametric Technology Corporation. All Rights Reserved.
User and training documentation from Parametric Technology Corporation (PTC) is subject to the copyright laws of the United States and other countries and is provided under a license agreement that restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the PTC copyright notice and any other proprietary notice provided by PTC. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of PTC and no authorization is granted to make copies for such purposes.

Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from PTC. UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION.

Registered Trademarks of Parametric Technology Corporation or a Subsidiary
Advanced Surface Design, Behavioral Modeling, CADDS, Computervision, CounterPart, Create · Collaborate · Control, EPD, EPD.Connect, Expert Machinist, Flexible Engineering, GRANITE, HARNESSDESIGN, Info*Engine, InPart, MECHANICA, Optegra, Parametric Technology, Parametric Technology Corporation, PartSpeak, PHOTORENDER, Pro/DESKTOP, Pro/E, Pro/ENGINEER, Pro/HELP, Pro/INTRALINK, Pro/MECHANICA, Pro/TOOLKIT, Product First, Product Development Means Business, Product Makes the Company, PTC, the PTC logo, PT/Products, Shaping Innovation, Simple · Powerful · Connected, The Way to Product First, and Windchill.

Trademarks of Parametric Technology Corporation or a Subsidiary
3DPAINT, Associative Topology Bus, AutobuildZ, CDRS, CV, CVact, CVaec, CVdesign, CV-DORS, CVMAC, CVNC, CVToolmaker, EDAcompare, EDAconduit, DataDoctor, DesignSuite, DIMENSION III, Distributed Services Manger, DIVISION, e/ENGINEER, eNC Explorer, Expert Framework, Expert MoldBase, Expert Toolmaker, FlexPDM, FlexPLM, Harmony, InterComm, InterComm Expert, InterComm EDAcompare, InterComm EDAconduit, ISSM, KDiP, Knowledge Discipline in Practice, Knowledge System Driver, ModelCHECK, MoldShop, NC Builder, POLYCAPP, Pro/ANIMATE, Pro/ASSEMBLY, Pro/CABLING, Pro/CASTING, Pro/CDT, Pro/CMM, Pro/COLLABORATE, Pro/COMPOSITE, Pro/CONCEPT, Pro/CONVERT, Pro/DATA for PDGS, Pro/DESIGNER, Pro/DETAIL, Pro/DIAGRAM, Pro/DIEFACE, Pro/DRAW, Pro/ECAD, Pro/ENGINE, Pro/FEATURE, Pro/FEM-POST, Pro/FICIENCY, Pro/FLY-THROUGH, Pro/HARNESS, Pro/INTERFACE, Pro/LANGUAGE, Pro/LEGACY, Pro/LIBRARYACCESS, Pro/MESH, Pro/Model.View, Pro/MOLDESIGN, Pro/NC-ADVANCED, Pro/NC-CHECK, Pro/NC-MILL, Pro/NC-POST, Pro/NC-SHEETMETAL, Pro/NC-TURN, Pro/NC-WEDM, Pro/NC-Wire EDM, Pro/NETWORK ANIMATOR, Pro/NOTEBOOK, Pro/PDM, Pro/PHOTORENDER, Pro/PIPING, Pro/PLASTIC ADVISOR, Pro/PLOT, Pro/POWER DESIGN, Pro/PROCESS, Pro/REPORT, Pro/REVIEW, Pro/SCAN-TOOLS, Pro/SHEETMETAL, Pro/SURFACE, Pro/VERIFY, Pro/Web.Link, Pro/Web.Publish, Pro/WELDING, ProductView, PTC Precision, Routed Systems Designer, Shrinkwrap, The Product Development Company, Validation Manager, Wildfire, Windchill DynamicDesignLink, Windchill PartsLink, Windchill PDMLink, Windchill ProjectLink, and Windchill SupplyLink.

Patents of Parametric Technology Corporation or a Subsidiary
Registration numbers and issue dates follow. Additionally, equivalent patents may be issued or pending outside of the United States. Contact PTC for further information.

GB2366639B 13-October-2004	GB2353376 05-November-2003	5,140,321 18-August-1992
GB2363208 25-August-2004	GB2354686 15-October-2003	5,423,023 05-June-1990
(EP/DE/GB)0812447 26-May-2004	6,545,671 B1 08-April-2003	4,310,615 21-December-1998
GB2365567 10-March-2004	GB2354685B 18-June-2003	4,310,614 30-April-1996

(GB)2388003B 21-January-2004	GB2354683B 04-June-2003	4,310,614 22-April-1999
6,665,569 B1 16-December-2003	6,608,623 B1 19-August-2003	5,297,053 22-March-1994
GB2353115 10-December-2003	6,473,673 B1 29-October-2002	5,513,316 30-April-1996
6,625,607 B1 23-September-2003	GB2354683B 04-June-2003	5,689,711 18-November-1997
6,580,428 B1 17-June-2003	6,447,223 B1 10-September-2002	5,506,950 09-April-1996
GB2354684B 02-July-2003	6,308,144 23-October-2001	5,428,772 27-June-1995
GB2384125 15-October-2003	5,680,523 21-October-1997	5,850,535 15-December-1998
GB2354096 12-November-2003	5,838,331 17-November-1998	5,557,176 09-November-1996
GB2354924 24-September-2003	4,956,771 11-September-1990	5,561,747 01-October-1996
6,608,623 B1 19-August-2003	5,058,000 15-October-1991	(EP)0240557 02-October-1986

Third-Party Trademarks

Adobe, Acrobat, Distiller, and the Acrobat logo are trademarks of Adobe Systems Incorporated. Advanced ClusterProven, ClusterProven, and the ClusterProven design are trademarks or registered trademarks of International Business Machines Corporation in the United States and other countries and are used under license. IBM Corporation does not warrant and is not responsible for the operation of this software product. AIX is a registered trademark of IBM Corporation. Allegro, Cadence, and Concept are registered trademarks of Cadence Design Systems, Inc. Apple, Mac, Mac OS, and Panther are trademarks or registered trademarks of Apple Computer, Inc. AutoCAD and Autodesk Inventor are registered trademarks of Autodesk, Inc. Baan is a registered trademark of Baan Company. CADAM and CATIA are registered trademarks of Dassault Systemes. COACH is a trademark of CADTRAIN, Inc. CYA, iArchive, HOTbackup, and Virtual StandBy are trademarks or registered trademarks of CYA Technologies, Inc. DOORS is a registered trademark of Telelogic AB. FLEXlm and FLEXnet are registered trademarks of Macrovision Corporation. Geomagic is a registered trademark of Raindrop Geomagic, Inc. EVERSYNC, GROOVE, GROOVEFEST, GROOVE.NET, GROOVE NETWORKS, iGROOVE, PEERWARE, and the interlocking circles logo are trademarks of Groove Networks, Inc. Helix is a trademark of Microcadam, Inc. HOOPS is a trademark of Tech Soft America, Inc. HP-UX is a registered trademark of Hewlett-Packard Company. I-DEAS, Metaphase, Parasolid, SHERPA, Solid Edge, TeamCenter, UG-NX, and Unigraphics are trademarks or registered trademarks of UGS Corp. InstallShield is a registered trademark and service mark of InstallShield Software Corporation in the United States and/or other countries. Intel is a registered trademark of Intel Corporation. IRIX is a registered trademark of Silicon Graphics, Inc. I-Run and ISOGEN are registered trademarks of Alias Ltd. LINUX is a registered trademark of Linus Torvalds. MainWin and Mainsoft are trademarks of Mainsoft Corporation. MatrixOne is a trademark of MatrixOne, Inc. Mentor Graphics and Board Station are registered trademarks and 3D Design, AMPLE, and Design Manager are trademarks of Mentor Graphics Corporation. MEDUSA and STHENO are trademarks of CAD Schroer GmbH. Microsoft, Microsoft Project, Windows, the Windows logo, Windows NT, Windows XP, Visual Basic, and the Visual Basic logo are registered trademarks of Microsoft Corporation in the United States and/or other countries. Moldflow is a registered trademark of Moldflow Corporation. Netscape and the Netscape N and Ship's Wheel logos are registered trademarks of Netscape Communications Corporation in the U.S. and other countries. Oracle is a registered trademark of Oracle Corporation. OrbixWeb is a registered trademark of IONA Technologies PLC. PDGS is a registered trademark of Ford Motor Company. RAND is a trademark of RAND Worldwide. Rational Rose is a registered trademark of Rational Software Corporation. RetrievalWare is a registered trademark of Convera Corporation. RosettaNet is a trademark and Partner Interface Process and PIP are registered trademarks of RosettaNet, a nonprofit organization. SAP and R/3 are registered trademarks of SAP AG Germany. SolidWorks is a registered trademark of SolidWorks Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and in other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun logo, Solaris, UltraSPARC, Java and all Java based marks, and "The Network is the Computer" are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and in other countries. 3Dconnexion is a registered trademark of Logitech International S.A. TIBCO is a registered trademark and TIBCO ActiveEnterprise, TIBCO Designer, TIBCO Enterprise Message Service, TIBCO Rendezvous, TIBCO TurboXML, and TIBCO BusinessWorks are the trademarks or registered trademarks of TIBCO Software Inc. in the United States and other countries. WebEx is a trademark of WebEx Communications, Inc.

Third-Party Technology Information

Certain PTC software products contain licensed third-party technology:

Rational Rose 2000E is copyrighted software of Rational Software Corporation.

RetrievalWare is copyrighted software of Convera Corporation.

VisTools library is copyrighted software of Visual Kinematics, Inc. (VKI) containing confidential trade secret information belonging to VKI.

HOOPS graphics system is a proprietary software product of, and is copyrighted by, Tech Soft America, Inc.

I-Run and ISOGEN are copyrighted software of Alias Ltd.

Xdriver is copyrighted software of 3Dconnexion, Inc, a Logitech International S.A. company.

G-POST is copyrighted software and a registered trademark of Intercim.

VERICUT is copyrighted software and a registered trademark of CGTech.

FLEXnet Publisher is copyrighted software of Macrovision Corporation.

Pro/PLASTIC ADVISOR is powered by Moldflow technology.

MainWin Dedicated Libraries are copyrighted software of Mainsoft Corporation.

DFORMD.DLL is copyrighted software from Compaq Computer Corporation and may not be distributed.

LightWork Libraries are copyrighted by LightWork Design 1990-2001.

Visual Basic for Applications and Internet Explorer is copyrighted software of Microsoft Corporation.

Parasolid is © UGS Corp.

TECHNOMATIX is copyrighted software and contains proprietary information of Technomatix Technologies Ltd.

TIBCO ActiveEnterprise, TIBCO Designer, TIBCO Enterprise Message Service, TIBCO Rendezvous, TIBCO TurboXML, and TIBCO BusinessWorks are provided by TIBCO Software Inc.

Technology "Powered by Groove" is provided by Groove Networks, Inc.

Technology "Powered by WebEx" is provided by WebEx Communications, Inc.

Oracle 8i run-time, Oracle 9i run-time, and Oracle 10g run-time are Copyright © 2002-2004 Oracle Corporation. Oracle programs provided herein are subject to a restricted use license and can only be used in conjunction with the PTC software they are provided with.

Adobe Acrobat Reader and Adobe Distiller are copyrighted software of Adobe Systems Inc. and are subject to the Adobe End-User License Agreement as provided by Adobe with those products.

METIS, developed by George Karypis and Vipin Kumar at the University of Minnesota, can be researched at http://www.cs.umn.edu/~karypis/metis. METIS is © 1997 Regents of the University of Minnesota.

Windchill Info*Engine Server contains IBM XML Parser for Java Edition and the IBM Lotus XSL Edition.

Pop-up calendar components Copyright © 1998 Netscape Communications Corporation. All Rights Reserved.

Apache Server, Tomcat, Xalan, Xerces, and Jakarta are technologies developed by, and are copyrighted software of, the Apache Software Foundation (http://www.apache.org) – their use is subject to the terms and limitations at: http://www.apache.org.

UnZip (© 1990-2001 Info-ZIP, All Rights Reserved) is provided "AS IS" and WITHOUT WARRANTY OF ANY KIND. For the complete Info-ZIP license see http://www.info-zip.org/doc/LICENSE.

The Java™ Telnet Applet (StatusPeer.java, TelnetIO.java, TelnetWrapper.java, TimedOutException.java), Copyright © 1996, 97 Mattias L. Jugel, Marcus Meißner, is redistributed under the GNU General Public License. This license is from the original copyright holder and the Applet is provided WITHOUT WARRANTY OF ANY KIND. You may obtain a copy of the source code for the Applet at http://www.mud.de/se/jta (for a charge of no more than the cost of physically performing the source distribution), by sending e-mail to leo@mud.de or marcus@mud.de—you are allowed to choose either distribution method. Said source code is likewise provided under the GNU General Public License.

GTK+ - The GIMP Toolkit is licensed under the GNU Library General Public License (LGPL). You may obtain a copy of the source code at http://www.gtk.org, which is likewise provided under the GNU LGPL.

zlib software Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

OmniORB is distributed under the terms and conditions of the GNU General Public License and GNU Library General Public License.

The Java Getopt.jar file, copyright 1987-1997 Free Software Foundation, Inc.

Java Port copyright 1998 by Aaron M. Renn (arenn@urbanophile.com), is redistributed under the GNU LGPL. You may obtain a copy of the source code at http://www.urbanophile.com/arenn/hacking/download.html. The source code is likewise provided under the GNU LGPL.

CUP Parser Generator Copyright ©1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian–used by permission. The authors and their employers disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall the authors or their employers be liable for any special, indirect or consequential damages, or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action arising out of or in connection with the use or performance of this software.

This product may include software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org): Copyright © 1998-2003 The OpenSSL Project. All rights reserved. This product may include cryptographic software written by Eric Young (eay@cryptsoft.com).

ImageMagick software is Copyright © 1999-2005 ImageMagick Studio LLC, a nonprofit organization dedicated to making software imaging solutions freely available. ImageMagick is freely available without charge and provided pursuant to the following license agreement: http://www.imagemagick.org/script/license.php.

Gecko and Mozilla components are subject to the Mozilla Public License Version 1.1 at http://www.mozilla.org/MPL. Software distributed under the Mozilla Public License (MPL) is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either expressed or implied. See the MPL for the specific language governing rights and limitations.

Mozilla Japanese localization components are subject to the Netscape Public License Version 1.1 (at http://www.mozilla.org/NPL). Software distributed under the Netscape Public License (NPL) is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either expressed or implied (see the NPL for the rights and limitations that are governing different languages). The Original Code is Mozilla Communicator client code, released March 31, 1998 and the Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright © 1998 Netscape Communications Corporation. All Rights Reserved. Contributors: Kazu Yamamoto (kazu@mozilla.gr.jp), Ryoichi Furukawa (furu@mozilla.gr.jp), Tsukasa Maruyama (mal@mozilla.gr.jp), Teiji Matsuba (matsuba@dream.com).

iCal4j is Copyright © 2005, Ben Fortuna, All rights reserved. Redistribution and use of iCal4j in source and binary forms, with or without modification, are permitted provided that the following conditions are met: (i) Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer; (ii) Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution; and (iii) Neither the name of Ben Fortuna nor the names of any other contributors may be used to endorse or promote products derived from this software without specific prior written permission. iCal4j SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Software may contain the Independent JPEG Group's JPEG software. This software is Copyright © 1991-1998, Thomas G. Lane. All Rights Reserved. This software is based in part on the work of the Independent JPEG Group.

Software may contain libpng, Copyright © 2004 Glenn Randers-Pehrson, which is distributed according to the disclaimer and license (as well as the list of Contributing Authors) at http://www.libpng.org/pub/png/src/libpng-LICENSE.txt.

UNITED STATES GOVERNMENT RESTRICTED RIGHTS LEGEND
This document and the software described herein are Commercial Computer Documentation and Software, pursuant to FAR 12.212(a)-(b) (OCT'95) or DFARS 227.7202-1(a) and 227.7202-3(a) (JUN'95), and are provided to the US Government under a limited commercial license only. For procurements predating the above clauses, use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 (OCT'88) or Commercial Computer Software-Restricted Rights at FAR 52.227-19(c)(1)-(2) (JUN'87), as applicable. 090805

Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494 USA

Table of Contents

Preface

Book Conventions _____ xlviii
Window Managers and the User Interface _____ xlix
Online User Documentation _____ xlix
Online Command Help _____ l
Printing Documentation _____ l
Resources and Services _____ li
Documentation Comments _____ li

CVact Overview

Introduction to CVact _____ 1-2
Description of CVact _____ 1-2
Advanced Customization Tool _____ 1-2
Designing a GUI _____ 1-2
Dialog Boxes _____ 1-2
Who Can Use CVact _____ 1-3
What Interface Designers Can Do _____ 1-3
What End Users Can Do _____ 1-3
Porting Custom Menus from UNIX to Windows _____ 1-3
What System Managers Can Do _____ 1-4
Basics of Creating an Interface _____ 1-4
Task Definition _____ 1-4
Graphic Objects _____ 1-4
Writing Callbacks _____ 1-5
Adding Application Code _____ 1-5

Sending Commands to CADDS 1-5
Creating a Shared Library 1-5
Bringing Up Your Interface from CADDS 1-6
Distributing Your CR Process 1-6
Getting Started 1-7
The File Pulldown 1-10
The Edit Pulldown 1-10
The View Pulldown 1-11
The Interfaces Pulldown 1-12
The Options Pulldown 1-12
The Mode Pulldown 1-13
The Tools Pulldown 1-13
The Selected Objects Menu 1-13
Using the Mouse 1-14
Exiting CVact 1-15
Objects Overview 1-16
Descriptions 1-16

Beginning Layouts

Creating Objects from the Palette 2-2
Creating a Command Argument Box (CAB) 2-2
Adding More Objects 2-3
Rearranging Objects in a CAB 2-4
Reparenting Objects 2-4
Tree of Objects 2-4
Using the Object Property Editor to Rearrange Objects 2-7
Changing the Object to be Edited in the Property Editor 2-9
Selecting Multiple Objects 2-9
Editing Object Properties 2-10
Setting Attributes in the Object Property Editor 2-10
The Message and Callback Editors 2-10
The Text Editor 2-11
Using Option Lists 2-12
Recognizing What Has Changed 2-13
Editing Properties of Multiple Objects 2-13

Saving the Interfaces ____ 2-14
Saving Your Dialog Box ____ 2-14
Using Your Interfaces Using CVact ____ 2-15
Removing Interfaces ____ 2-15

Changing Your Interface

Hierarchies ____ 3-2
Parenting Hierarchy ____ 3-2
Instancing Hierarchy ____ 3-2
Managing Geometry and Layout ____ 3-3
Geometry Management ____ 3-3
Visibility ____ 3-4
Sensitivity ____ 3-5
Dependencies ____ 3-6
Dependency Expressions ____ 3-7
Setting Dependency Attributes ____ 3-8
Specifying Layout Options ____ 3-10
Specifying and Using Icons ____ 3-11
Icon Examples ____ 3-12
Icon Editor ____ 3-14
Icon Converter ____ 3-16
Syntax ____ 3-16
Description ____ 3-16
Example ____ 3-16
Callbacks ____ 3-17
Types of Callbacks ____ 3-17
Using Simple Callbacks ____ 3-20
Arguments Passed to the Callback ____ 3-20
thisObj ____ 3-22
systemData ____ 3-22
clientData ____ 3-22
HLIString Manipulation Utilities ____ 3-22
Using the UI Callback Editor ____ 3-23

Creating a Simple Dialog Box ____ 3-24
Customizing Objects ____ 3-30
Customizing Most Attributes ____ 3-30
Resources ____ 3-30
Sensitivity and Visibility ____ 3-30
Color and Font Category ____ 3-31
Attributes of Font Categories ____ 3-33
Attributes of Colors ____ 3-34

Executing Your Menus

Generating Shared Libraries ____ 4-2
Creating Shared Libraries ____ 4-2
Standalone Testing Using CVUI (supplied under /usr/apl/cvact/bin) ____ 4-3
Preparing to Use Your Menus in CADDS ____ 4-3
Using Your Menus in CADDS ____ 4-4
Platform Specific Information ____ 4-5
Using CVact Menus with CADDS on Windows ____ 4-7
Dynamically Loading CVact Menus ____ 4-8
Using the Sample Menu ____ 4-9
The Sample Menu ____ 4-9
Browsing Through the Sample Menu ____ 4-10
Geometry Management ____ 4-10
Using Editors and Submenus ____ 4-10
Specifying Initial and Final Codes for .act Files ____ 4-10
Customizing Your Menus ____ 4-12
Sample of the .CR file ____ 4-15
How to Override Customization ____ 4-15
Example of Overriding Design Rule Customization ____ 4-15
Behavior of CVact Menus when Iconized or Deiconized ____ 4-16
Indicating the State of CADDS Desktop ____ 4-17

CVact Attributes

Attribute List ____ 5-2

Partial Syntax 5-2
AbortCb 5-2
Accelerator cust 5-3
ActivateCb 5-3
ActiveWindow 5-3
Alignment 5-3
AlternateIcon cust 5-4
ApplyButton 5-4
ApplyCb 5-4
ApplyLabel cust
ApplyLabelFromMessage
ApplyLabelFromVMessage 5-5
ArgumentBox 5-5
AssistLine cust
AssistLineFromMessage
AssistLineFromVMessage 5-5
AssistOn cust 5-6
Background 5-6
ButtonParent 5-6
CancelCb 5-6
CancelLabel
CancelLabelFromMessage
CancelLabelFromVMessage 5-7
Cell 5-7
CellValues 5-7
Checked cust 5-7
Children 5-8
ChildrenNum 5-8
Chosen 5-8
ChosenCb 5-8
ChosenItems 5-9
ChosenItemsNum 5-9
ChosenObject 5-9
ClosedCb 5-9
Column Alignments 5-10
ColumnLabel 5-10
ColumnLabels 5-10
ColumnWidths 5-10
ColumnsNum 5-10
ColumnsWide 5-11
CommandOn cust 5-11

CompleteCb 5-11
CompleteCellCb 5-12
ConstantHeight 5-12
ConstantWidth 5-13
Contents cust 5-13
ContentsNum 5-13
CurrentTool cust 5-13
Cursor 5-13
CursorPosition 5-14
DeactivateCb 5-14
DecimalPlaces cust 5-14
DefaultActionCb 5-14
DefaultBoolean cust 5-15
DefaultButton 5-15
DefaultFloatValue cust 5-15
DefaultIntegerValue cust 5-15
DefaultItems 5-16
DefaultItemsNum 5-16
DefaultObject 5-16
DefaultString cust
DefaultStringFromMessage
DefaultStringFromVMessage 5-16
DefaultStringNum 5-17
DefaultValue cust 5-17
Defective 5-17
Definer 5-17
Dependency 5-17
DependencyExpression 5-18
DependencyType 5-18
DestroyCb 5-18
DropCb 5-18
Editor 5-19
EditorAttributes 5-19
EditorFocusChangeCb 5-19
EditorIcon 5-19
EditorInvokedCb 5-20
EditorRescindCb 5-20
Entry 5-20
FloatValue 5-20
FontCategory 5-21
Foreground 5-21

Framed ____ 5-21
Gutter ____ 5-21
Height ____ 5-22
HeightSpecified ____ 5-22
HelpContext ____ 5-22
HLS cust ____ 5-23
Icon cust ____ 5-23
IconifiedIcon cust ____ 5-23
IconifiedTitlecust
IconifiedTitleFromMessage
IconifiedTitleFromVMessage ____ 5-23
InitializeCb ____ 5-24
Inline cust ____ 5-24
InputPermitted ____ 5-24
IntegerValue ____ 5-25
IsEditableCb ____ 5-25
IsInputOkCb ____ 5-25
Item ____ 5-26
ItemsNum ____ 5-26
LabelFontCategory ____ 5-26
Manage ____ 5-26
MaxFloatValue cust ____ 5-27
MaxIntegerValue cust ____ 5-27
MaxLength ____ 5-27
MaxVisibleLength ____ 5-27
MaxVisibleLines ____ 5-28
MinFloatValue cust ____ 5-28
MinIntegerValue cust ____ 5-28
MinVisibleColumns ____ 5-28
MinVisibleLength ____ 5-29
MinVisibleLines ____ 5-29
MinVisibleRows ____ 5-29
Modal ____ 5-29
MouseEventCb ____ 5-30
MouseEventMask ____ 5-30
Name ____ 5-30
NumColumns cust ____ 5-31
OkLabelcust
OkLabelFromMessage
OkLabelFromVMessage ____ 5-31
Orientation ____ 5-31

OtherData ____ 5-31
Parent ____ 5-32
PercentDrawn ____ 5-32
Pointer ____ 5-32
PositionInDesktop ____ 5-32
Prefilled ____ 5-32
Prompt
PromptFromMessage
PromptFromVMessage ____ 5-33
PromptOn cust ____ 5-33
PropertiesCb ____ 5-33
PushedCb ____ 5-33
RGB ____ 5-34
RadioBehavior ____ 5-34
Remembered ____ 5-34
RememberedBoolean ____ 5-34
RememberedItems ____ 5-35
RememberedItemsNum ____ 5-35
RememberedObject ____ 5-35
RememberedString ____ 5-35
RememberedStringNum ____ 5-35
ResetCb ____ 5-36
ResetLabel
ResetLabelFromMessage
ResetLabelFromVMessage ____ 5-36
RowColumn ____ 5-36
RowLabel ____ 5-37
RowLabels ____ 5-37
RowsHigh ____ 5-37
RowsNum ____ 5-37
Sensitive cust ____ 5-37
SensitiveNow ____ 5-38
Source cust ____ 5-38
Specifier cust ____ 5-38
StartColumn ____ 5-39
StartRow ____ 5-39
State ____ 5-39
StatusArea ____ 5-39
Sticky ____ 5-39
TextFontCategory ____ 5-40
TextString

TextStringFromMessage
TextStringFromVMessage ____ 5-40
TextStringNum ____ 5-40
TextValue cust
TextValueFromMessage
TextValueFromVMessage ____ 5-40
Title cust
TitleFromMessage
TitleFromVMessage ____ 5-41
UserText cust ____ 5-41
Visible ____ 5-41
VisibleNow ____ 5-42
VisibleWhenManaged cust ____ 5-42
Width ____ 5-42
WidthSpecified ____ 5-43
WindowActivateCb ____ 5-43
WindowTitle cust
WindowTitleFromMessage
WindowTitleFromVMessage ____ 5-43
XPosition ____ 5-44
XSpecified ____ 5-44
YPosition ____ 5-44
YSpecified ____ 5-45

CVact Classes

Hierarchy of Class Attributes and UI Functions ____ 6-3
How to Use the Inheritance Tree ____ 6-3

ActionEntry ____ 6-8
Purpose ____ 6-8
User Interactions ____ 6-8

AligningForm ____ 6-9
Purpose ____ 6-9
Description ____ 6-9

ApplicationDesktop ____ 6-11
Purpose ____ 6-11
Description ____ 6-11
Constraints ____ 6-11

ApplyPulldown _____ 6-12
Purpose _____ 6-12
Description _____ 6-12
Constraints _____ 6-12
CommandBox (CAB) _____ 6-14
Purpose _____ 6-14
Description _____ 6-14
User Interactions _____ 6-14
Constraints _____ 6-14
CommandEntry _____ 6-16
Purpose _____ 6-16
Description _____ 6-16
Container _____ 6-17
Purpose _____ 6-17
Description _____ 6-17
Constraints _____ 6-17
DataSetWindow _____ 6-18
Purpose _____ 6-18
Description _____ 6-18
Constraints _____ 6-18
DialogBox _____ 6-19
Purpose _____ 6-19
Description _____ 6-19
Constraints _____ 6-19
DirectActionEntry _____ 6-20
Purpose _____ 6-20
Description _____ 6-20
EditorDialogBox _____ 6-21
Purpose _____ 6-21
Description _____ 6-21
Associating the EditorDialogBox with a Fillin Field _____ 6-21
Entry _____ 6-22
Purpose _____ 6-22
Description _____ 6-22

EntryList 6-23
Purpose 6-23
Description 6-23
ErrorBox 6-24
Purpose 6-24
Description 6-24
FailureBox 6-25
Purpose 6-25
Description 6-25
Fillin 6-26
Purpose 6-26
Description 6-26
Constraints 6-26
FloatFillin 6-28
Purpose 6-28
Description 6-28
Constraints 6-28
IdentificationBox 6-30
Purpose 6-30
Description 6-30
InformationBox 6-31
Purpose 6-31
Description 6-31
IntegerFillin 6-32
Purpose 6-32
Description 6-32
Constraints 6-33
Label 6-34
Purpose 6-34
Description 6-34
NonAligningForm 6-35
Purpose 6-35
Description 6-35
Constraints 6-35

OptionList 6-36
Purpose 6-36
Description 6-36
Palette 6-37
Purpose 6-37
Description 6-37
Paragraph 6-38
Purpose 6-38
Description 6-38
Constraints 6-39
PushButton 6-40
Purpose 6-40
Description 6-40
Constraints 6-40
Usage Tips 6-40
QueryResultBox 6-42
Purpose 6-42
Description 6-42
QuestionBox 6-43
Purpose 6-43
Description 6-43
RadioSet 6-44
Purpose 6-44
Description 6-44
ScrollList 6-45
Purpose 6-45
Description 6-45
ScrollMatrix 6-47
Purpose 6-47
Description 6-47
Separator 6-49
Purpose 6-49
Description 6-49
Constraints 6-49

SlideOffEditorDlgBox ____ 6-50
Purpose ____ 6-50
Description ____ 6-50
Associating the SlideOffEditorDlgBox with a Fillin Field ____ 6-50
Slider ____ 6-52
Purpose ____ 6-52
Description ____ 6-52
Constraints ____ 6-52
SpecifierEntry ____ 6-53
Purpose ____ 6-53
Description ____ 6-53
Toggle ____ 6-54
Purpose ____ 6-54
Description ____ 6-54
ToggleEntry ____ 6-55
Purpose ____ 6-55
Description ____ 6-55
TopLevel ____ 6-56
Purpose ____ 6-56
Description ____ 6-56
WarningBox ____ 6-57
Purpose ____ 6-57
Description ____ 6-57
WorkingBox ____ 6-58
Purpose ____ 6-58
Description ____ 6-58

Ui Functions

Attributes ____ A-6
UiActivate ____ A-10
Syntax ____ A-10
UiAddColumns ____ A-11

Purpose ____ A-11
Syntax ____ A-11
Description ____ A-11
Example ____ A-11

UiAddEntry ____ A-12

Purpose ____ A-12
Syntax ____ A-12
Description ____ A-12
Example ____ A-12

UiAddItem ____ A-13

Purpose ____ A-13
Syntax ____ A-13
Description ____ A-13
Example ____ A-13
Example 2 ____ A-13
Notes ____ A-14
See Also ____ A-14

UiAddItems ____ A-15

Purpose ____ A-15
Syntax ____ A-15
Description ____ A-15
Example ____ A-15
Notes ____ A-16
See Also ____ A-16

UiAddRows ____ A-17

Purpose ____ A-17
Syntax ____ A-17
Description ____ A-17
Example ____ A-17
Notes ____ A-18

UiAddSeparator ____ A-19

Purpose ____ A-19
Syntax ____ A-19
Description ____ A-19
Example ____ A-19

UiAligningFormCreate ____ A-20

Purpose ____ A-20
Syntax ____ A-20
Description ____ A-20
Example ____ A-20
Notes ____ A-20

UiAppDtCreate ____ A-21
Purpose ____ A-21
Syntax ____ A-21
Description ____ A-21
Example ____ A-21

UiApplicationStateCreate ____ A-22
Purpose ____ A-22
Syntax ____ A-22
Description ____ A-22
Notes ____ A-22

UiApplyPulldownCreate ____ A-23
Purpose ____ A-23
Syntax ____ A-23
Description ____ A-23
Example ____ A-23
Notes ____ A-23

UiAssistOn ____ A-24
Purpose ____ A-24
Syntax ____ A-24
Description ____ A-24
Example ____ A-24
Notes ____ A-24
See Also ____ A-24

UiChooseItem ____ A-25
Purpose ____ A-25
Syntax ____ A-25
Description ____ A-25
Example ____ A-25
Notes ____ A-25

UiClose ____ A-26
Purpose ____ A-26

Syntax _____ A-26
Description _____ A-26
Example _____ A-26
Notes _____ A-26

UiCommandBoxCreate _____ A-27

Purpose _____ A-27
Syntax _____ A-27
Description _____ A-27
Example _____ A-27
Notes _____ A-28

UiCommandEntryCreate _____ A-29

Purpose _____ A-29
Syntax _____ A-29
Description _____ A-29
Example _____ A-29

UiCompletion _____ A-31

Purpose _____ A-31
Syntax _____ A-31
Description _____ A-31
Example _____ A-31

UiConfirm _____ A-32

Purpose _____ A-32
Syntax _____ A-32
Description _____ A-32
Example _____ A-32

Ui...CreateFromVStatus _____ A-33

Purpose _____ A-33
Syntax _____ A-33
Description _____ A-33
Notes _____ A-33

UiDatasetWindowCreate _____ A-34

Purpose _____ A-34
Syntax _____ A-34
Description _____ A-34
Example _____ A-34

UiDeleteColumns _____ A-35

Purpose ____ A-35
Syntax ____ A-35
Description ____ A-35

UiDeleteEntry ____ A-36
Purpose ____ A-36
Syntax ____ A-36
Description ____ A-36
Example ____ A-36
See Also ____ A-36

UiDeleteItem ____ A-37
Purpose ____ A-37
Syntax ____ A-37
Description ____ A-37
Example ____ A-37
See Also ____ A-37

UiDeleteRows ____ A-38
Purpose ____ A-38
Syntax ____ A-38
Description ____ A-38

UiDeleteSeparator ____ A-39
Purpose ____ A-39
Syntax ____ A-39
Description ____ A-39
Example ____ A-39
See Also ____ A-39

UiDestroy ____ A-40
Purpose ____ A-40
Syntax ____ A-40
Description ____ A-40

UiDialogBoxCreate ____ A-41
Purpose ____ A-41
Syntax ____ A-41
Description ____ A-41
Example ____ A-41

UiDirectEntryCreate ____ A-42

Purpose _____ A-42
Syntax _____ A-42
Description _____ A-42
Notes _____ A-42

UiEditCell _____ A-43

Purpose _____ A-43
Syntax _____ A-43
Description _____ A-43

UiEditorDialogBoxCreate _____ A-44

Purpose _____ A-44
Syntax _____ A-44

UiEndBusy _____ A-45

Purpose _____ A-45
Syntax _____ A-45
Description _____ A-45
Example _____ A-45
See Also _____ A-45

UiEntryListCreate _____ A-46

Purpose _____ A-46
Syntax _____ A-46

UiErrorBoxCreate
UiErrorBoxCreateFromStatus
UiErrorBoxCreateFromVStatus _____ A-47

Purpose _____ A-47
Syntax _____ A-47
Description _____ A-47

UiFailureBoxCreate _____ A-48

Purpose _____ A-48
Syntax _____ A-48
Description _____ A-48

UiFillinCreate _____ A-49

Purpose _____ A-49
Syntax _____ A-49
Description _____ A-49

UiFillinStrlen _____ A-50

Purpose ____ A-50
Syntax ____ A-50
Description ____ A-50
Notes ____ A-50

UiFloatFillinCreate ____ A-51
Purpose ____ A-51
Syntax ____ A-51
Description ____ A-51

UiGet... ____ A-52
Purpose ____ A-52
Syntax ____ A-52
Description ____ A-52
See Also ____ A-52

UiGet...Cb ____ A-53
Purpose ____ A-53
Syntax ____ A-53
Description ____ A-53
Notes ____ A-54

UiGetCell ____ A-55
Purpose ____ A-55
Syntax ____ A-55
Description ____ A-55

UiGetCellColor ____ A-56
Purpose ____ A-56
Syntax ____ A-56
Description ____ A-56
Notes ____ A-56

UiGetColumnColors ____ A-57
Purpose ____ A-57
Syntax ____ A-57
Description ____ A-57

UiGetColumnLabel ____ A-58
Purpose ____ A-58
Syntax ____ A-58
Description ____ A-58

See Also ____ A-58
UiGetDependency ____ A-59
Purpose ____ A-59
Syntax ____ A-59
See Also ____ A-59
UiGetEntry ____ A-60
Purpose ____ A-60
Syntax ____ A-60
Description ____ A-60
UiGetEntryLabel ____ A-61
Purpose ____ A-61
Syntax ____ A-61
Description ____ A-61
UiGetEntryRequired ____ A-62
Purpose ____ A-62
Syntax ____ A-62
Description ____ A-62
UiGetFocus ____ A-63
Purpose ____ A-63
Syntax ____ A-63
Description ____ A-63
Example ____ A-63
Notes ____ A-63
UiGetGeometry ____ A-64
Purpose ____ A-64
Syntax ____ A-64
Description ____ A-64
UiGetGeometrySpecified ____ A-65
Purpose ____ A-65
Syntax ____ A-65
Description ____ A-65
UiGetInstanceId ____ A-66
Purpose ____ A-66
Syntax ____ A-66
Description ____ A-66

Example ____ A-66
See Also ____ A-66

UiGetItem ____ A-67
Purpose ____ A-67
Syntax ____ A-67
Description ____ A-67
See Also ____ A-67

UiGetItemSensitive ____ A-68
Purpose ____ A-68
Syntax ____ A-68
Description ____ A-68

UiGetObjectByName ____ A-69
Purpose ____ A-69
Syntax ____ A-69
Description ____ A-69

UiGetPositionInDesktop ____ A-71
Purpose ____ A-71
Syntax ____ A-71
Description ____ A-71

UiGetResetInProgress ____ A-72
Purpose ____ A-72
Syntax ____ A-72
Description ____ A-72

UiGetRowColors ____ A-73
Purpose ____ A-73
Syntax ____ A-73
Description ____ A-73

UiGetRowColumn ____ A-74
Syntax ____ A-74

UiGetRowLabel ____ A-75
Purpose ____ A-75
Syntax ____ A-75
Description ____ A-75

UiGetRowLabels ____ A-76

Purpose ____ A-76
Syntax ____ A-76
Description ____ A-76

UilIdentificationBoxCreate ____ A-77
Purpose ____ A-77
Syntax ____ A-77

UilInformationBoxCreate
UilInformationBoxCreateFromStatus
UilInformationBoxCreateFromVStatus ____ A-78
Purpose ____ A-78
Syntax ____ A-78
Description ____ A-78
Notes ____ A-78

UilIntegerFillinCreate ____ A-79
Purpose ____ A-79
Syntax ____ A-79
Description ____ A-79

UilIsDefaultButton ____ A-80
Purpose ____ A-80
Syntax ____ A-80
Description ____ A-80

UiLabelCreate ____ A-81
Purpose ____ A-81
Syntax ____ A-81
Description ____ A-81
Example ____ A-81

UiNonAligningFormCreate ____ A-82
Purpose ____ A-82
Syntax ____ A-82
Description ____ A-82
Example ____ A-82

UiObtainEntry ____ A-83
Purpose ____ A-83
Syntax ____ A-83
Description ____ A-83

UiOptionListCreate ____ A-84
Purpose ____ A-84
Syntax ____ A-84
Description ____ A-84
UiPaletteCreate ____ A-85
Purpose ____ A-85
Syntax ____ A-85
Description ____ A-85
UiParagraphCreate ____ A-86
Purpose ____ A-86
Syntax ____ A-86
Description ____ A-86
UiParagraphInsertText ____ A-87
Purpose ____ A-87
Syntax ____ A-87
Description ____ A-87
Example ____ A-87
Notes ____ A-87
UiParagraphMoveCursor ____ A-88
Purpose ____ A-88
Syntax ____ A-88
Description ____ A-88
Example ____ A-88
Notes ____ A-88
UiPostCab ____ A-89
Purpose ____ A-89
Syntax ____ A-89
Description ____ A-89
Example ____ A-89
UiPromptOn ____ A-90
Syntax ____ A-90
UiPushButtonCreate ____ A-91
Purpose ____ A-91
Syntax ____ A-91
Example ____ A-91

UiQueryItem ____ A-92
Purpose ____ A-92
Syntax ____ A-92
Description ____ A-92
Example ____ A-92
Notes ____ A-92
UiQueryItmStatusByIdx ____ A-93
Purpose ____ A-93
Syntax ____ A-93
Description ____ A-93
Example ____ A-93
Notes ____ A-93
UiQueryResultBoxCreate ____ A-94
Purpose ____ A-94
Syntax ____ A-94
Description ____ A-94
UiQuestionBoxCreate
UiQuestionBoxCreateFromStatus
UiQuestionBoxCreateFromVStatus ____ A-95
Purpose ____ A-95
Syntax ____ A-95
Description ____ A-95
Notes ____ A-95
UiRadioScrollListCreate ____ A-96
Purpose ____ A-96
Syntax ____ A-96
Description ____ A-96
Notes ____ A-96
UiRadioSetCreate ____ A-97
Purpose ____ A-97
Syntax ____ A-97
Example ____ A-97
Notes ____ A-97
See Also ____ A-97
UiReset ____ A-98
Purpose ____ A-98

Syntax ____ A-98
Description ____ A-98
Notes ____ A-98

UiResetCab ____ A-99
Purpose ____ A-99
Syntax ____ A-99
Description ____ A-99
Example ____ A-99
Notes ____ A-99

UiScrollListCreate ____ A-100
Purpose ____ A-100
Syntax ____ A-100
Description ____ A-100
Example ____ A-100

UiScrollMatrixCreate ____ A-101
Purpose ____ A-101
Syntax ____ A-101
Description ____ A-101

UiSeparatorCreate ____ A-102
Purpose ____ A-102
Syntax ____ A-102

UiSet... ____ A-103
Purpose ____ A-103
Syntax ____ A-103
Description ____ A-103
See Also ____ A-103

UiSetBorderWidthForSldoffDlgBox ____ A-104
Purpose ____ A-104
Syntax ____ A-104
Description ____ A-104
Example ____ A-104
Notes ____ A-104

UiSet...Cb ____ A-105
Purpose ____ A-105
Syntax ____ A-105

Description A-105
Notes A-106

UiSetCell A-107

Purpose A-107
Syntax A-107
Description A-107
Example A-107
Notes A-107

UiSetCellColor A-108

Purpose A-108
Syntax A-108
Description A-108
Notes A-108

UiSetCellValues A-109

Purpose A-109
Syntax A-109
Description A-109
Example A-109
Notes A-109

UiSetChosen A-110

Purpose A-110
Syntax A-110
Description A-110
Example A-110
Notes A-110

UiSetColumnAlignments A-111

Purpose A-111
Syntax A-111
Description A-111
Example A-111

UiSetColumnColors A-112

Purpose A-112
Syntax A-112
Description A-112

UiSetColumnLabel A-113

Purpose ____ A-113
Syntax ____ A-113
Description ____ A-113

UiSetColumnLabels ____ A-114
Purpose ____ A-114
Syntax ____ A-114
Description ____ A-114
Example ____ A-114

UiSetDefaultButtonWithSThickness ____ A-115
Purpose ____ A-115
Syntax ____ A-115

UiSetDependency ____ A-116
Syntax ____ A-116
See Also ____ A-116

UiSetEditorAttributes ____ A-117
Syntax ____ A-117
See Also ____ A-117

UiSetEntryLabel ____ A-118
Purpose ____ A-118
Syntax ____ A-118
Description ____ A-118

UiSetEntryRequired ____ A-119
Purpose ____ A-119
Syntax ____ A-119
Description ____ A-119

UiSet...FromMessage
UiSet...FromVMessage ____ A-120
Purpose ____ A-120
Syntax ____ A-120
Textual Objects Set from Messages ____ A-120

UiSetGeometrySpecified ____ A-121
Purpose ____ A-121
Syntax ____ A-121
Description ____ A-121
See Also ____ A-121

UiSetInstanceId ____ A-122
Purpose ____ A-122
Syntax ____ A-122
Description ____ A-122
See Also ____ A-122
UiSetItemSensitive ____ A-123
Purpose ____ A-123
Syntax ____ A-123
Description ____ A-123
UiSetMatrix ____ A-124
Purpose ____ A-124
Syntax ____ A-124
Description ____ A-124
Example ____ A-124
Notes ____ A-125
UiSetRowColor ____ A-126
Purpose ____ A-126
Syntax ____ A-126
Description ____ A-126
UiSetRowColumn ____ A-127
Syntax ____ A-127
UiSetRowLabel ____ A-128
Purpose ____ A-128
Syntax ____ A-128
Description ____ A-128
UiSetRowLabels ____ A-129
Purpose ____ A-129
Syntax ____ A-129
Description ____ A-129
Example ____ A-129
UiSlideOffDialogBoxCreate ____ A-130
Purpose ____ A-130
Syntax ____ A-130
Description ____ A-130
UiSlideOffEditorDialogBoxCreate ____ A-131

Syntax ____ A-131

UiSetSpecifiedPosition ____ A-132

Purpose ____ A-132
Syntax ____ A-132
Description ____ A-132

UiSetSpecifiedSize ____ A-133

Purpose ____ A-133
Syntax ____ A-133
Description ____ A-133

UiSpecifierCreate ____ A-134

Purpose ____ A-134
Syntax ____ A-134

UiStartBusy ____ A-135

Purpose ____ A-135
Syntax ____ A-135
Description ____ A-135
Example ____ A-135
See Also ____ A-135

UiToggleCreate ____ A-136

Purpose ____ A-136
Syntax ____ A-136
Description ____ A-136
Example ____ A-136

UiToggleEntryCreate ____ A-137

Purpose ____ A-137
Syntax ____ A-137
Description ____ A-137
Notes ____ A-137

UiWarningBoxCreate
UiWarningBoxCreateFromStatus
UiWarningBoxCreateFromVStatus ____ A-138

Purpose ____ A-138
Syntax ____ A-138
Description ____ A-138

UiWorkingBoxCreate ____ A-139

Purpose _____ A-139
Syntax _____ A-139

Hli Functions

HliCCharToChar () _____ B-3
Syntax _____ B-3
Description _____ B-3
Example _____ B-3

HliCNStringToString () _____ B-4
Syntax _____ B-4
Description _____ B-4
Example _____ B-4
Notes _____ B-4

HliCStrdup () _____ B-5
Purpose _____ B-5
Syntax _____ B-5
Description _____ B-5
Example _____ B-5
Notes _____ B-5
See Also _____ B-5

HliCStringFromString () _____ B-6
Syntax _____ B-6
Description _____ B-6
Example _____ B-6
Notes _____ B-6
See Also _____ B-6

HliCStringToString () _____ B-7
Syntax _____ B-7
Example _____ B-7
Notes _____ B-7
See Also _____ B-7

HliCStrndup () _____ B-8
Syntax _____ B-8

HliCharToCChar () _____ B-9

Syntax ____ B-9
HliExternNStringToString () ____ B-10
Purpose ____ B-10
Syntax ____ B-10
Description ____ B-10
Notes ____ B-10
HliExternStringFromString () ____ B-11
Purpose ____ B-11
Syntax ____ B-11
Description ____ B-11
Notes ____ B-11
HliExternStringToString () ____ B-12
Purpose ____ B-12
Syntax ____ B-12
Description ____ B-12
Notes ____ B-12
HliFromCString () ____ B-13
Syntax ____ B-13
Notes ____ B-13
HliFromExternString () ____ B-14
Purpose ____ B-14
Syntax ____ B-14
Description ____ B-14
Notes ____ B-14
HliShutdown () ____ B-15
Syntax ____ B-15
HliStringNToCString () ____ B-16
Purpose ____ B-16
Syntax ____ B-16
Description ____ B-16
Notes ____ B-16
HliStringNToExternString () ____ B-17
Purpose ____ B-17
Syntax ____ B-17
Description ____ B-17

HliStringToCString () B-18
Syntax B-18
HliStringToExternString () B-19
Purpose B-19
Syntax B-19
Description B-19
HliStringVFormat () B-20
Syntax B-20

CL Functions

Notes on CL Functions C-4
Sending Commands to CADDS C-4
Invoking a CR Function from CADDS C-5
Public Interfaces to CADDS Functions C-6
Sending Coding Error Messages To CADDS C-7
Using Active Variables between CADDS, CV-DORS, and CR Process C-7
Active Variable Types C-8
CL Bindings for Active Variables from CADDS C-8
CL Bindings for Active Variables from CR C-9
ClAbortCommand C-10
Purpose C-10
Syntax C-10
Notes C-10
ClAssistOn C-11
Purpose C-11
Syntax C-11
Example C-11
Notes C-11
ClAVAddCallback C-12
Purpose C-12
Syntax C-12
Description C-12
Example C-12
Notes C-13

See Also ____ C-13
ClAVCreate ____ C-14
Purpose ____ C-14
Syntax ____ C-14
Description ____ C-14
Example ____ C-14
Notes ____ C-14
See Also ____ C-15
ClAVGetDoubleVal ____ C-16
Purpose ____ C-16
Syntax ____ C-16
Description ____ C-16
Example ____ C-16
Notes ____ C-16
See Also ____ C-16
ClAVGetFloatVal ____ C-17
Purpose ____ C-17
Syntax ____ C-17
Description ____ C-17
Example ____ C-17
Notes ____ C-17
See Also ____ C-17
ClAVGetHliStringVal ____ C-18
Purpose ____ C-18
Syntax ____ C-18
Description ____ C-18
Example ____ C-18
Notes ____ C-18
See Also ____ C-19
ClAVGetIntegerVal ____ C-20
Purpose ____ C-20
Syntax ____ C-20
Description ____ C-20
Example ____ C-20
Notes ____ C-20
See Also ____ C-20

ClAVGetStringVal _____ C-21
Purpose _____ C-21
Syntax _____ C-21
Description _____ C-21
Example _____ C-21
Notes _____ C-21
See Also _____ C-22
ClAVGetValue _____ C-23
Purpose _____ C-23
Syntax _____ C-23
Description _____ C-23
Example _____ C-23
Notes _____ C-24
ClAVRemoveCallback _____ C-25
Purpose _____ C-25
Syntax _____ C-25
Description _____ C-25
See Also _____ C-25
ClAVSetDoubleVal _____ C-26
Purpose _____ C-26
Syntax _____ C-26
Description _____ C-26
See Also _____ C-26
ClAVSetFloatVal _____ C-27
Purpose _____ C-27
Syntax _____ C-27
Description _____ C-27
Notes _____ C-27
See Also _____ C-27
ClAVSetHliStringVal _____ C-28
Purpose _____ C-28
Syntax _____ C-28
Description _____ C-28
See Also _____ C-28
ClAVSetIntegerVal _____ C-29
Purpose _____ C-29

Syntax ____ C-29
Description ____ C-29
See Also ____ C-29

ClAVSetStringVal ____ C-30
Purpose ____ C-30
Syntax ____ C-30
Description ____ C-30
See Also ____ C-30

ClAVSetValue ____ C-31
Purpose ____ C-31
Syntax ____ C-31
Description ____ C-31
Notes ____ C-31

ClBufferCCommand ____ C-32
Purpose ____ C-32
Syntax ____ C-32
Description ____ C-32
Example ____ C-32
Notes ____ C-34

ClBufferMsg ____ C-35
Syntax ____ C-35
Description ____ C-35
Notes ____ C-35

ClCaddsRtlInstanceWrap ____ C-36
Purpose ____ C-36
Syntax ____ C-36

ClCaddsRtlSizeWrap ____ C-37
Purpose ____ C-37
Syntax ____ C-37

Cl_call_CR_func ____ C-38
Purpose ____ C-38
Syntax ____ C-38
Description ____ C-38
Examples ____ C-39
Notes ____ C-40

ClClearPrompt _____ C-41
Purpose _____ C-41
Syntax _____ C-41
Cl_E_epsilon_value _____ C-42
Purpose _____ C-42
Syntax _____ C-42
Cl_E_extents_value _____ C-43
Purpose _____ C-43
Syntax _____ C-43
ClFlushCommand _____ C-44
Purpose _____ C-44
Syntax _____ C-44
Notes _____ C-44
ClGetDialogInstances _____ C-45
Purpose _____ C-45
Syntax _____ C-45
Description _____ C-45
ClGetDialogLastInstanceId _____ C-46
Purpose _____ C-46
Syntax _____ C-46
Description _____ C-46
See Also _____ C-46
ClGetDialogMaxInstances _____ C-47
Purpose _____ C-47
Syntax _____ C-47
Description _____ C-47
See Also _____ C-47
ClGetDialogNumCurInstances _____ C-48
Purpose _____ C-48
Syntax _____ C-48
Description _____ C-48
See Also _____ C-48
ClGetFloatStringVal _____ C-49
Purpose _____ C-49
Syntax _____ C-49

Notes _____ C-49
CIGetIntegerStringVal _____ C-50
Purpose _____ C-50
Syntax _____ C-50
Notes _____ C-50
CIInitialize _____ C-51
Purpose _____ C-51
Syntax _____ C-51
Example _____ C-51
CILaunchDialog _____ C-52
Purpose _____ C-52
Syntax _____ C-52
CIRegisterDialog _____ C-53
Purpose _____ C-53
Syntax _____ C-53
Description _____ C-53
CIRegisterDialogMaxInstances _____ C-54
Purpose _____ C-54
Syntax _____ C-54
Description _____ C-54
Example _____ C-55
See Also _____ C-55
CIRequest _____ C-56
Purpose _____ C-56
Syntax _____ C-56
Description _____ C-56
CI_RTL_caddspaths_size _____ C-57
Purpose _____ C-57
Syntax _____ C-57
CI_RTL_e_drawing_name_init _____ C-58
Purpose _____ C-58
Syntax _____ C-58
Example _____ C-58
Notes _____ C-58

Cl_RTL_e_drawing_name_instance C-59
Purpose C-59
Syntax C-59
Example C-59
Cl_RTL_e_drawing_name_size C-60
Purpose C-60
Syntax C-60
Example C-60
Notes C-60
Cl_RTL_e_nfig_all_size C-61
Purpose C-61
Syntax C-61
Cl_RTL_e_tag_all_size C-62
Purpose C-62
Syntax C-62
Cl_RTL_e_view_name_size C-63
Purpose C-63
Syntax C-63
Cl_RTL_caddspaths_get_name C-64
Purpose C-64
Syntax C-64
Description C-64
Notes C-64
ClSendCCommand C-65
Purpose C-65
Syntax C-65
Description C-65
Notes C-65
ClSendErrorCode C-66
Purpose C-66
Syntax C-66
ClSendErrorCString C-67
Purpose C-67
Syntax C-67

CISendErrorHliString _____ C-68
Purpose _____ C-68
Syntax _____ C-68
CISendWarning _____ C-69
Purpose _____ C-69
Syntax _____ C-69
CISetPrompt _____ C-70
Purpose _____ C-70
Syntax _____ C-70
Example _____ C-70

Glossary

Preface

CVact User Guide describes how to use the menus, editors, component files (such as icon and message repositories), utilities, and other design features of CVact. CVact was originally developed for people with backgrounds in computer-aided geometric modeling who wanted to develop new CADDS property sheets or applications that support CADDS look and feel.

Ideally, you can be an interface designer and not a programmer. This manual is also for you if you intend to write callbacks and application code that work with the interface objects. CVact is a GUI builder that has two parts:

- An IDT, or Interactive Design Tool, that lets you develop interfaces on the screen
- Libraries of user interface components appropriate for programmers to access

This manual explains how to use the IDT part of CVact. To use both the IDT and the program libraries, you should:

- Be familiar with CADDS task sets and property sheets
- Be able to use a development environment suitable for C
- Have access to training materials sufficient to program in C.

CVact User Guide also explains:

- The general concept and process of interfacing with CADDS
- How to design interfaces
- Changing a layout and object properties
- Changing your interface's icons and text
- Using the test mode in CVact
- Running some of the utility programs associated with CVact

We wish to acknowledge the contribution of Ms. D'lorah A. Hunt, winner of the 1994 Best Paper - Software Track, CVUG International Conference, to earlier editions of this User Guide.

Book Conventions

The following table illustrates and explains conventions used in writing about CADDS applications.

Convention	Example	Explanation
Menu selections and options	List Section option, Specify Layer field	Indicates a selection you must make from a menu or property sheet or a text field that you must fill in.
User-selected graphic location	X, d_1 or P1	Marks a location or entity selection in graphic examples.
User input in CADDS text fields and on any command line	`cvaec.hd.data.param` `tar -xvf /dev/rst0`	Enter the text in a CADDS text field or on any command line.
System output	`Binary transfer complete.`	Indicates system responses in the CADDS text window or on any command line.
Variable in user input	`tar -cvf /dev/rst0` filename	Replace the variable with an appropriate substitute; for example, replace filename with an actual file name.
Variable in text	tagname	Indicates a variable that requires an appropriate substitute when used in a real operation; for example, replace tagname with an actual tag name.
CADDS commands and modifiers	INSERT LINE TANTO	Shows CADDS commands and modifiers as they appear in the command line interface.
Text string	`"SRFGROUPA"` or `'SRFGROUPA'`	Shows text strings. You must enclose text string with single or double quotation marks.
Integer	n	Supply an integer for the *n.*
Real number	x	Supply a real number for the *x*.
#	`# mkdir /cdrom`	Indicates the root (superuser) prompt on command lines.
%	`% rlogin` remote_system_name `-l root`	Indicates the C shell prompt on command lines.
$	`$ rlogin` remote_system_name `-l root`	Indicates the Bourne shell prompt on command lines.

Window Managers and the User Interface

According to the window manager that you use, the look and feel of the user interface in CADDS can change. Refer to the following table:

Look and Feel of User Interface Elements

User Interface Element	Common Desktop Environment (CDE) on Solaris, HP, and IBM	Window Manager Other Than CDE on Solaris, HP, IBM, and Windows
Option button	ON — Round, filled in the center OFF — Round, empty	ON — Diamond, filled OFF — Diamond, empty
Toggle key	ON — Square with a check mark OFF — Square, empty	ON — Square, filled OFF — Square, empty

Online User Documentation

Online documentation for each book is provided in HTML if the documentation CD-ROM is installed. You can view the online documentation in the following ways:

- From an HTML browser
- From the Information Access button on the CADDS desktop or the Local Data Manager (LDM)

Please note: The LDM is valid only for standalone CADDS.

You can also view the online documentation directly from the CD-ROM without installing it.

From an HTML Browser:

1. Navigate to the directory where the documents are installed. For example,

   `/usr/apl/cadds/data/html/htmldoc/` (UNIX)

   Drive:`\usr\apl\cadds\data\html\htmldoc\` (Windows)
2. Click `mainmenu.html`. A list of available CADDS documentation appears.
3. Click the book title you want to view.

From the Information Access Button on the CADDS Desktop or LDM:

1. Start CADDS.
2. Choose Information Access, the i button, in the top-left corner of the CADDS desktop or the LDM.
3. Choose DOCUMENTATION. A list of available CADDS documentation appears.
4. Click the book title you want to view.

From the Documentation CD-ROM:

1. Mount the documentation CD-ROM.
2. Point your browser to:

   CDROM_mount_point/htmldoc/mainmenu.html (UNIX)

   CDROM_Drive:\htmldoc\mainmenu.html (Windows)

Online Command Help

You can view the online command help directly from the CADDS desktop in the following ways:

- From the Information Access button on the CADDS desktop or the LDM
- From the command line

From the Information Access Button on the CADDS Desktop or LDM:

1. Start CADDS.
2. Choose Information Access, the i button, in the top-left corner of the CADDS desktop or the LDM.
3. Choose COMMAND HELP. The Command Help property sheet opens displaying a list of verb-noun combinations of commands.

From the Command Line: Type the exclamation mark (!) to display online documentation before typing the verb-noun combination as follows:

```
#01#!INSERT LINE
```

Printing Documentation

A PDF (Portable Document Format) file is included on the CD-ROM for each online book. See the first page of each online book for the document number referenced in the PDF file name. Check with your system administrator if you need more information.

You must have Acrobat Reader installed to view and print PDF files.

The default documentation directories are:

- /usr/apl/cadds/data/html/pdf/doc_number.pdf (UNIX)
- CDROM_Drive:\usr\apl\cadds\data\html\pdf\doc_number.pdf (Windows)

Resources and Services

For resources and services to help you with PTC (Parametric Technology Corporation) software products, see the *PTC Customer Service Guide*. It includes instructions for using the World Wide Web or fax transmissions for customer support.

Documentation Comments

PTC welcomes your suggestions and comments. You can send feedback electronically to `doc-webhelp@ptc.com`.

Chapter 1

CVact Overview

This chapter provides an overview to the CVact tool. In it you find information relating to designing and building a graphical user interface, description of the CVact tool and of the objects available in CVact.

- Introduction to CVact
- Getting Started
- Objects Overview

Introduction to CVact

Description of CVact

CVact is a comprehensive, object-oriented, graphical user interface (GUI) builder for developing high-quality GUIs supporting both CADDS and Motif look and feel. It provides you, the software developer, with the drag-and-drop tools to interactively create, modify, test, and generate code for the user interface portion of your applications.

Advanced Customization Tool

CVact's Advanced Customization Tool contains a palette of objects. You can drag-and-drop these objects to interactively create, modify, test, and generate code for the user interface portion of your application. Because you can modify interfaces while the underlying application is connected and running, you can immediately see the effect on your application. You can then test your interface, automatically compile it as C++ code and link it to any UNIX based application(s).

Designing a GUI

When you design your interface, you can customize the appearance and behavior of any object you selected from the Palette:UiObjects window and placed in your interface, by using the Property Editor window. The Property Editor window allows you to change the default object properties. Then to get an overall view, you can see the tree structure in the Advanced Customizer window as you are inserting objects.

Dialog Boxes

Using CVact, you can create dialog boxes with a CADDS LAF (Look and Feel) similar in style to CADDS menus. These menus are Motif compliant and can be viewed both in CADDS LAF and Motif LAF. Dialog boxes are useful especially for new CADDS users, because the dialog boxes remember:

- Complex command syntax; you supply just the individual parameters
- Previously entered information
- Default information
- Application specific information
- End user modification

Who Can Use CVact

As a toolkit, CVact is developed to be used by the user interface designer and programmer for the purpose of building interfaces for end users of application products. To use CVact, you must know basic C programming and have a working knowledge of the UNIX environment.

The resulting interfaces and dialog boxes contain components that can be modified by the end user.

What Interface Designers Can Do

As a qualified user interface designer, you can use CVact to lay out the components of an interface to build an effective GUI to an application product. As such, you can use CVact to:

- Define the layout and appearance of an interface
- Present graphic UI objects to gather input from users
- Define what happens when push buttons are selected, fields are completed and check boxes are checked
- Integrate the interface with your application, and with other applications, such as CADDS.

What End Users Can Do

As an end user, you can further customize the dialog boxes and menu entries so that your version of the application is tailored to your needs.

To accomplish this, you can remove fields, delete options from option lists, change defaults, edit labels, and freeze field contents at selected values, among other things.

Porting Custom Menus from UNIX to Windows

The custom CVact menus are dynamically loaded on Windows for use with CADDS. However, you cannot use CVact interactive tool on Windows. You can copy the menus that you have designed on UNIX with CVact onto Windows. With CADDS on Windows, you can compile and use the CVact menus, message repositories, and XPM (X Pixmap or the XPM 2C format) icons that you have designed on UNIX and use them with CADDS.

What System Managers Can Do

As a system manager, you can customize the interface for your group or department by modifying the X Resource file.

Basics of Creating an Interface

This section presents some of the basic steps involved in the process of creating an interface.

You can use CVact to create dialog boxes and include them with an existing CADDS menu. Or you can use CVact with non-CADDS applications to give them a similar look and feel to CADDS.

In addition to creating a graphical user interface, you can program the callbacks to make analysis and design rule decisions. For example, you can program the menu such that when a value is entered for a corner radius, calculations are performed which select the correct tool for manufacturing and the process for CVNC machining.

Task Definition

Before beginning to use CVact, you must define the task which will be accomplished by using the interface. The task might be to create a custom dialog box allowing your end users to interact with CADDS in a fashion that is tailored for them. Or the task might be to allow your end users to create a frequently used graphic entity, such as a circle.

Graphic Objects

Once you have decided on the task, the next step is to identify the objects that correspond to the type of data, whether it is a text string, a real number or integer, a yes/no question, a discreet set of values and so on. Elsewhere in this book is a brief description of the various objects that CVact makes available.

Figure 1-1 Graphical Objects

The menu was created using three components:

- A Command Argument Box, which is a special category of dialog box, automatically contains the push buttons for Apply, Reset, and Cancel, the window title (which was changed to read “hello”), and the thumb tack.
- A label, which reads “Enter some text.”
- A fillin field.

Writing Callbacks

Although you can specify some interaction between the various interface components by entering attribute settings for them, most of the work involved in programming an interface occurs when you program the response to an end user action.

Instructions for using CVact to write callbacks, which are C functions that follow user actions, are included later in this book.

Adding Application Code

When writing callbacks that respond to user input, you are creating the feel of the menu.

If you are writing a very simple application, such as adding two positive numbers, you might want to validate that the numbers are positive. Your application code is written to do this. We have included some code fragments or examples to help you start. These files are located under `/usr/apl/cvact/examples/CODE`.

Sending Commands to CADDS

One function of CVact is to customize CADDS applications. There are CL (CADDS library) functions available that allow you to send commands to the CADDS command line.

Creating a Shared Library

After you have designed your interface, created your callbacks and application code, and tested your interface using the test mode of CVact, you can create a shared library.

The user interface in CADDS can use this shared library as well as provided executables. Using shared libraries, users can easily and quickly create and modify their menus.

This procedure automatically creates `Makefile` and other supporting C++ files, compiles the routines and creates a library called `libUserCr.so` on Solaris platforms. On other supported platforms, the library will be named appropriately. This library could be run in standalone mode through the CVUI command or with the CR while running CADDS.

Bringing Up Your Interface from CADDS

To use your menus while in CADDS, you must have the directory containing your `libUserCr.so` in the `LD_LIBRARY_PATH` before `/usr/apl/cvact/slib` and `/usr/apl/cadds/slib`.

Distributing Your CR Process

All the necessary files to run your menus are provided with CADDS. If you are using CVact in a standalone application, you are required to distribute other shared libraries. You can use the UNIX commands `ldd` or `cv_ldd` to find the missing libraries:

```
# cv_ldd `which CVUI`
```

Getting Started

To start CVact:

1. Change to the working directory, for example:

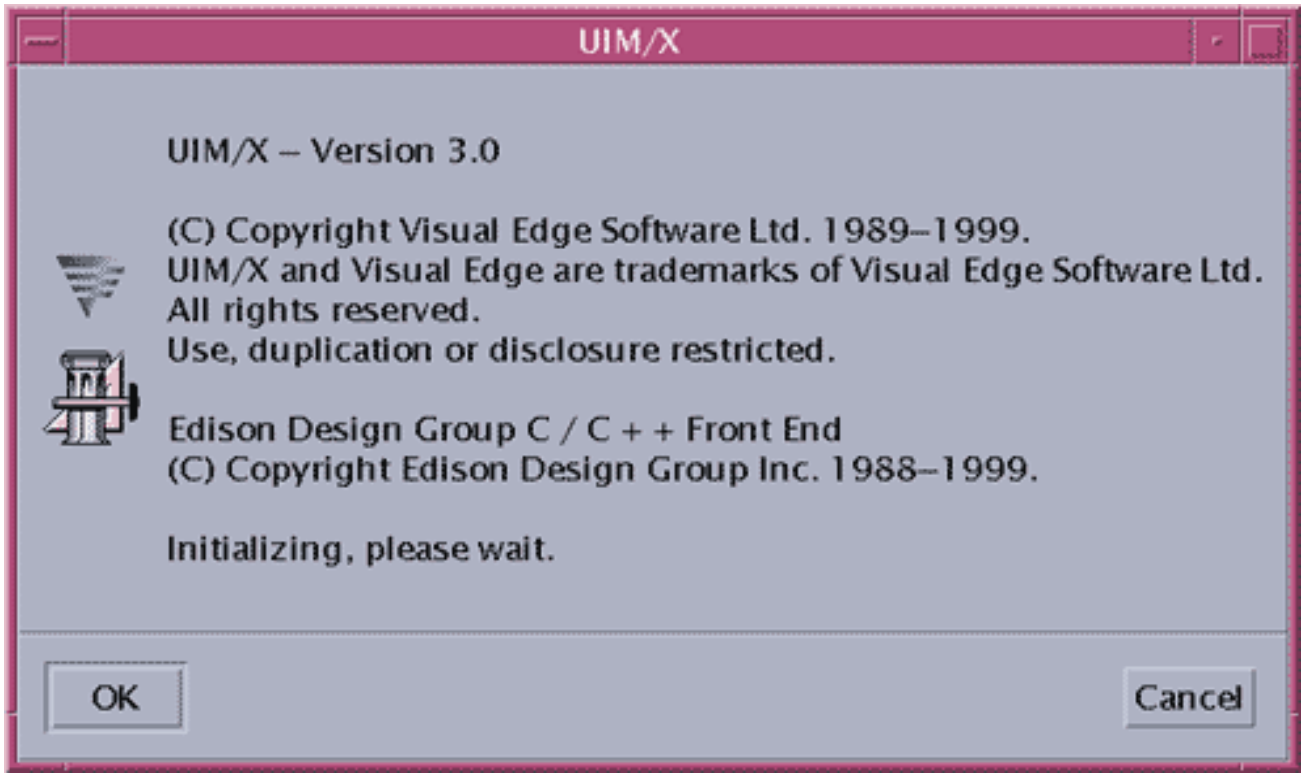
```
# cd /users/cvuser/menus/ins_cir
```

2. Enter the name of the program:

```
# cvact
```

The CVact Trademark window appears.

Figure 1-2 CVact Trademark Window

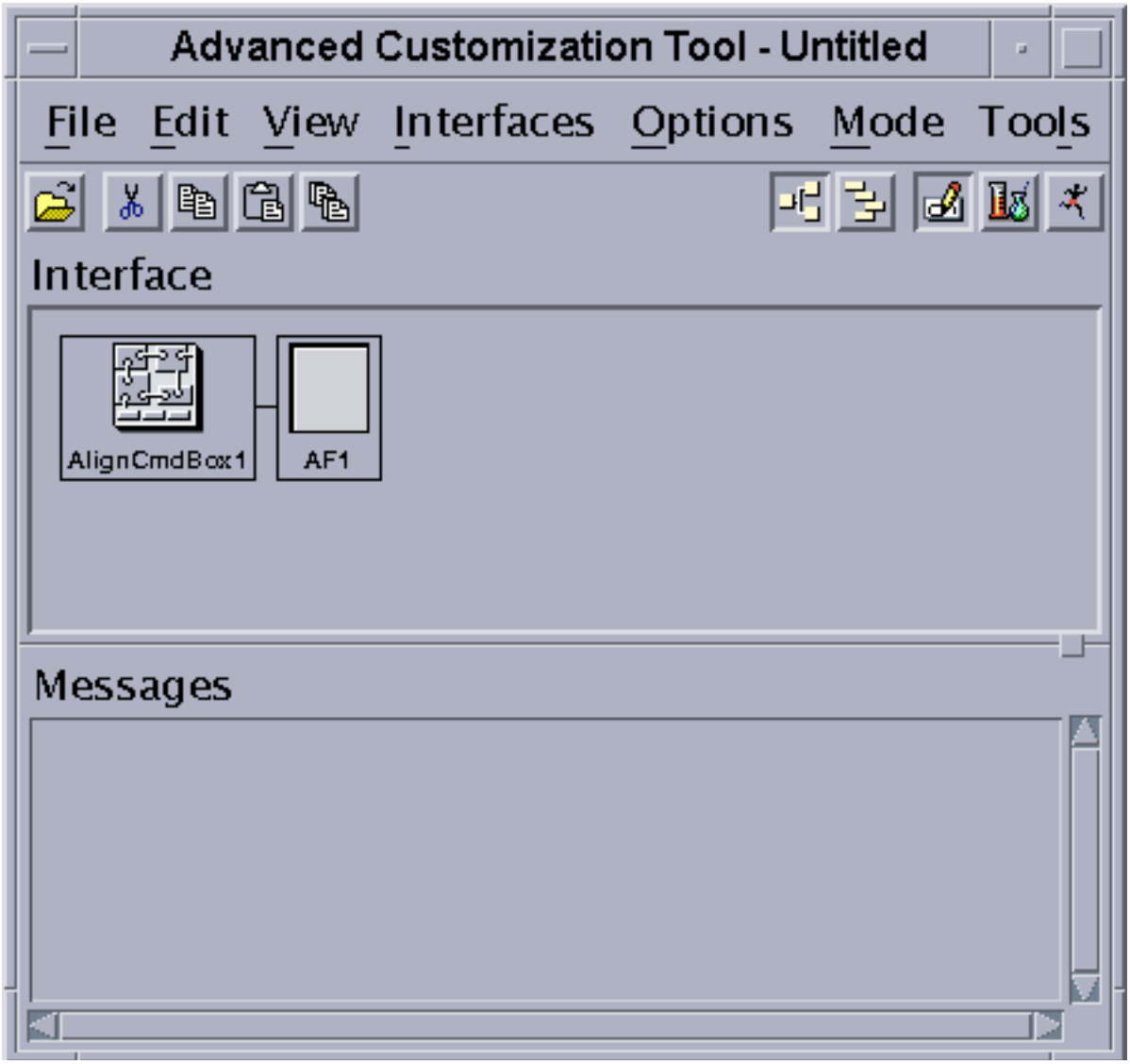

UIM/X
UIM/X – Version 3.0
(C) Copyright Visual Edge Software Ltd. 1989–1999.
UIM/X and Visual Edge are trademarks of Visual Edge Software Ltd.
All rights reserved.
Use, duplication or disclosure restricted.
Edison Design Group C / C + + Front End
(C) Copyright Edison Design Group Inc. 1988–1999.
Initializing, please wait.
OK
Cancel

The Advanced Customization Tool window and the Palette: UIObjects window automatically replaces this window. The object hierarchy area of the Advanced Customization Tool window displays labeled boxes for each object in an hierarchical tree format as you add it to your interface.

The message area displays system messages, including error messages, and output from using the test mode.

The menu bar allows you to access pulldown menus so that you can issue commands.

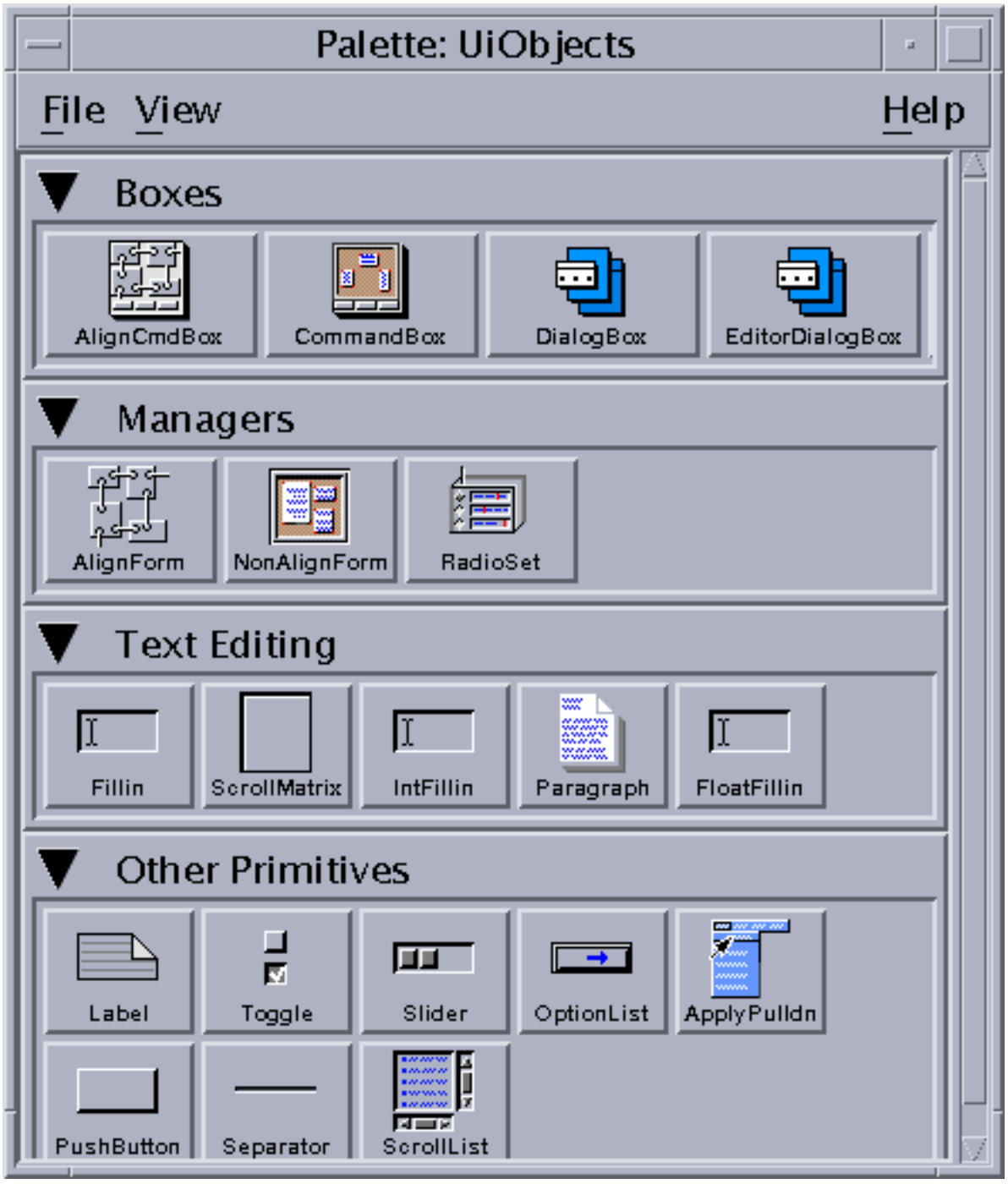
Figure 1-3 Advanced Customization Tool Window



The two toggles labeled Design and Test allow you to choose the mode of use, either Design mode to create and modify interfaces or Test mode to enable your interface to respond to user input. On startup, the default mode is Design mode. Use Test mode to demonstrate the functionality of your menu.

While in Design mode, you create user interfaces directly on your desktop by selecting a top level object from the Palette: UIObjects window and placing that object on your desktop. You then select other objects and place them within the top level object.

Figure 1-4 UIObjects Palette



While in Test mode, the Palette: UIObjects window is removed, and your interface responds to your user input, such as pressing a toggle or entering data into a fill in field. Output that your interface sends to your application is displayed in the message area of the Advanced Customization Tool window.

The File Pulldown

The File pulldown from the menu bar provides entries for opening and saving interfaces, resetting CVact, and so on, as shown in the following table.

Table 1-1 The File Pulldown

Entry	Action
Open	Opens a file to load an interface.
Reset	Resets the design tool. This is the same as exiting and re-entering the design tool.
Save Interface As...	Saves the current interface to a specified act file.
Exit	Exits the CVact tool.

The Edit Pulldown

The Edit pulldown lets you copy, cut, paste, and delete objects, in addition to opening the Object Property and Declarations editors, as listed in the following table.

Table 1-2 The Edit Pulldown

Entry	Action
Cut	Cuts the selected object. If the object has children, those objects are also cut.
Copy	Copies the selected object. If the object has children, those objects are also copied.
Paste	Pastes a previously cut or copied object onto a selected object.
Paste Before	Pastes a previously cut or copied object before the selected object.
Paste After	Pastes a previously cut or copied object after the selected object.
Duplicate	Duplicates the selected object. If the object has children, those objects are also duplicated.
Arrange	Displays the Arrange subment.See Table 1-3 on page 11.
Delete	Deletes the selected object. If the object has children, those objects are also deleted.
Other...	Displays the Other submenu. See Table 1-4 on page 11.
Select All	Selects all the objects.
Deselect All	Removes the selection from all the objects.
Clear Messages	Clears all the messages.

The table below displays entries that let you modify the arrangement of the objects as desired.

Table 1-3 Arrange Submenu

Entry	Action
	Places the objects at the vertical center.
	Makes the vertical spacing between the objects uniform.
	Arranges the objects along a vertical line
	Places the objects at the horizontal center.
	Makes the horizontal spacing between the objects uniform.
	Arranges the objects along a horizontal line.

The table below displays entries that let you move, resize, and duplicate objects, in addition to redisplaying the interface.

Table 1-4 Other Submenu

Entry	Action
Recreate	Recreates the interface should CVact be unable to display it or have hidden it.
Move	Moves the selected object. The pointer changes to a four-directional arrow pointer.
Resize	Resizes the selected object. The pointer changes to a two-directional arrow pointer.

The View Pulldown

The View pulldown gives you some control over whether the interface is shown as an outline or a tree and whether the objects' class names are shown. It also allows you select or deselect all objects at once.

Table 1-5 View Menu

Entry	Action
Show Interface	Displays the interface.

Table 1-5 View Menu

Entry	Action
Hide Interface	Hides the interface
Expand Node	Expands the selected tree node.
Contract Node	Contracts the selected tree node.
Tree	Displays the interface in the object hierarchy area in a tree format.
Outline	Displays the interface in the object hierarchy area in an outline format.
By Name	Displays only the names on the buttons.
By Icon	Displays only the icons on the buttons.
By Name and Icon	Displays both name and icon on each button.
Show Class	Displays the name of the class for each object in the interface.
Browser Area	Displays the browser area.
Messages Area	Displays the messages area.
Hide All Areas	Hides both the browser area and the messages area.

The Interfaces Pulldown

The Interface pulldown lists the currently loaded interfaces. With this pulldown, you can toggle between various interfaces.

The Options Pulldown

The Options pulldown provides the following menu items.

Table 1-6 Options Pulldown

Entry	Action
Current Directory...	Changes the current directory. The current directory is the directory from which you started CVact.
Arrange Spacing	Enables you to specify the horizontal and vertical spacing between the objects.
Bubble Help	Enables you to activate or deactivate bubble help and to set the delay time for displaying bubble help.
Property Editor...	Displays a Property Editor window. See the next table.
Save Options...	Sets the name of the file for the design tool default options.

The Property Editor sets the defaults for the Property Editor by displaying the following items in a submenu.

Table 1-7 Property Editor Submenu

Entry	Action
Hide Defaults	Hide properties which have default values in the property editor.
Hide Not Equal	

The Mode Pulldown

The Mode pulldown provides the following menu items.

Table 1-8 Mode Pulldown

Entry	Action
Design	Creates the menus
Test	Tests the menus
Run	Runs the executable by defining all the executables.

The Tools Pulldown

The Tools pulldown provides the following menu items.

Table 1-9 Tools Pulldown

Entry	Action
Property Editor	Displays a Property Editor window.
Connection Editor	Displays a Connection Editor window.
Declaration Editor	Displays a Declaration Editor window.
Interpreter...	Displays the Interpreter window.
System Palette...	Displays the Palette:UIObjects window.

The Selected Objects Menu

The Selected Objects menu is displayed when you press the MENU button while in the Advanced Customization Tool window. The menu items for this popup menu are as follows:

Table 1-10 Selected Objects Menu

Entry	Action
Tools	Displays the Tools submenu. See Table 1-9 on page 13.
Expand Node	Expands the selected tree node.
Contract Node	Contracts the selected tree node.

Table 1-10 Selected Objects Menu

Entry	Action
Cut	Cuts the selected object. If the object has children, those objects are also cut.
Copy	Copies the selected object. If the object has children, those objects are also copied.
Paste	Pastes a previously cut or copied object onto a selected object.
Paste Before	Pastes a previously cut or copied object before the selected object.
Paste After	Pastes a previously cut or copied object after the selected object.
Duplicate	Duplicates the selected object. If the object has children, those objects are also duplicated.
Arrange	Displays the Arrange submenu.
Delete	Deletes the selected object. If the object has children, those objects are also deletes.
Other...	Displays the Other submenu. See Table 1-4 on page 1-11.
Select All	Selects all the nodes.
Deselect All	Deselects all the nodes.

Using the Mouse

You use the mouse to perform many of the design tool operations. This manual occasionally refers to the mouse buttons as SELECT, ADJUST, and MENU. The buttons on your mouse might be configured differently, but the default assignments for them are listed below.

Table 1-11 Mouse Button Table

Button	Position on a three-button mouse	Position on a two-button mouse
SELECT	Left	Left
ADJUST	Middle	Left + Right
MENU	Right	Right

This mouse operations are:

Table 1-12 Mouse Button Actions

Action	Description
Click	Press and immediately release a button.
Press	Press a mouse button and continue to press it until instructed to release it.
Drag	Press a button while simultaneously moving the mouse.
Drop	Release the button at the end of a drag.

To access the options on the design tool menu bar, use the SELECT button. To access the design tool popup menus, use the MENU button.

Exiting CVact

To exit CVact, select

- File -> Exit to exit without saving your work, or
- File -> Save Interface As to save your current work.

Objects Overview

Descriptions

The following icons are graphic representations of the objects available in CVact:

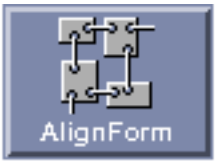
The Command Argument Box (CAB), seen as `AlignCmdBox`, contains a titlebar, a thumbtack, three push buttons (Apply, Cancel, and Reset), and an aligning form. A CAB can only be a parent object.

A Command Box (`CommandBox`) is similar to a CAB, except that it contains a nonaligned form; x and y locations must be specified. Typically you use a Command Box to hold interface objects that must always remain in the same relative position.

A Dialog Box (`DialogBox`) is a frame for holding other objects. It is similar to a CAB, except that it contains no thumbtack, push buttons or aligning form. You can use it to create task sets and other dialog boxes that do not close or open during the life of the application and do not require an Apply function.

An Editor Dialog Box (`EditorDialogBox`) allows you to specify an `editor` that can be used to provide input or verification for specific fields. The CADDS calculator is an example of a such an editor.

A `SlideOffEditorDialogBox` exhibits a slide off behavior. It allows you to fill out or edit the contents of a fillin field. It is useful for interactive designing and should be used programmatically.



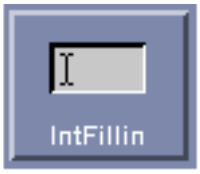
An Aligning Form (AF) seen as `AlignForm`, automatically aligns any children objects that are placed in it. The AF rearranges them in rows and columns adjusting to the space they need and according to its own size. An AlignForm can contain other Aligning Forms and Non Aligning Forms.

A Non Aligning Form (NAF) seen as `NonAlignForm`, does not do an automatic realignment of its children. You can use an NAF when you do not want the

objects aligned visually into rows and columns. An NAF can contain other alignment forms.

A `RadioSet` is a container like `aligning form` which can contain any child object as well as a set of toggles with radio behavior, only one of which can be selected at any time (see toggles). When you click one, it turns off the other one. You might use a radio set when a user should choose one option from a small set of choices.

`Fillin` fields come in three varieties: `Fillin`, which lets a user enter a single line of text input; `IntegerFillin`, for integer fields that accept an integer value; and `FloatFillin`, for floating point numbers. Application programs can do automatic verification of the contents. Both `IntegerFillin` and `FloatFillin` come with built-in calculator.



An `IntegerFillin` is a text fillin field with an editor button. A `calculator` is also provided with it. The `IntegerFillin` allows you to enter single-line text of an integer value directly into the text field using normal fillin field text entry methods or using the calculator. You can also perform the Inverse functions of Sin, Cos and Tan using the `calculator`.

A `FloatFillin` is a text fillin field displayed with an editor button that leads to a floating-point `calculator`. The `FloatFillin` allows you to enter single-line text of a floating point value either directly into the text field or the `calculator`.

A `ScrollMatrix`, otherwise known as a `scroll matrix`, is a spread-sheet like table of `Fillin` cells that can contain numbers or text. The user can scroll the table from cell to cell. You might use a scroll matrix for tabular information.

A `paragraph` allows scrolling, word-wrapped input. You may use a paragraph when you want your user to type in notes for something, or when you want to display more than a single line in a dialog box.

A `Label` can display characters or an icon. It is typically used to label fillin fields or provide other information in a dialog box.

A `Toggle` is a push button that can be either on or off. It is useful when you want your user to indicate “yes” or “no” or ON/OFF for a particular option. Toggles in a radio set have radio behavior, only one toggle can be selected at a time.

A `Slider` is an analog display/input device. The `Slider` allows you to enter a value by the slide operation with the direct manipulation of the mouse. It also enables both integer and float readouts through a scale. It can only be used programmatically for interactive design.

An `OptionList`, also called a `drop-down menu`, appears as a button with a textual description. You press the button to display other options. If you choose one of them, it becomes the label for the button until another option is selected.

A `ScrollList` is like an option list which has more options than can be displayed at one time. You might use a scroll list if you ask your user to choose a single item, or several items, from a given list.

An `ApplyPulldown`, otherwise known as an `apply pulldown`, is used to provide an extra level of interaction with the user after he or she has pressed the Apply button in a Command Argument Box. Typically this interaction involves more menu entries.

A PushButton (`push button`) can appear as either text or an icon or both. Typically, the user clicks on it to initiate some kind of action such as accepting the settings for a particular dialog box or accessing a different set of parameters within the same box.

A `Separator` is an object that is simply a line. It can be vertical or horizontal. You use a separator to separate different types of choices or input forms in a dialog box, or to otherwise draw attention to a set of objects that share something in common.

Chapter 2

Beginning Layouts

This chapter describes how to use the palette and the Advanced Customization Tool window to create an interface.

We provide a sample program that you may wish to refer to as you work with this manual.

This chapter will help you familiarize yourself with the types of objects that you can create. You can create simple dialog boxes to practice using the CVact tool and learn how to rearrange objects in an interface and to modify basic object properties.

- Creating Objects from the Palette
- Rearranging Objects in a CAB
- Editing Object Properties
- Saving the Interfaces

Creating Objects from the Palette

This section describes the basic mechanisms that you use to create interface objects.

Creating a Command Argument Box (CAB)

Follow the method given below to create a UI interface object container such as a dialog box:

1. Select AlignCmdBox from the Palette: UIObject window. (If you select CommandBox the children are not automatically aligned.)

Figure 2-1 AlignCmdBox

If the Palette window does not appear, verify that you are in the proper mode by selecting the Design toggle.

2. Move the object marker ( ) to the desired location for the upper left corner of the interface object.
3. Click the mouse button to fix the object at the chosen location.

A command argument box appears with a built-in Aligning Form.

Figure 2-2 The Command Argument Box

This box will hold the remaining interface objects needed to complete the task.

Adding More Objects

Once you have created a CAB, you can drop other objects into it. For example:

1. Select the Label object from the Palette: UIObjects window.
2. Move the object marker ( ) to a location within the CAB.
3. Click the mouse button to drop the Label.
4. Repeat the procedure for other objects.

Figure 2-3 Command Align Box

You can also use the Edit menu or the Selected Objects menu to access the copy, paste, and duplicate functions.

Rearranging Objects in a CAB

Once in the Command Argument Box, you can move objects. The two methods by which you can rearrange objects are:

- Reparenting by which you can place objects inside a container, parent, or objects.
- Changing the row and column specifications of the container objects.

Reparenting Objects

You can select objects to hold other objects. When the object is selected, it becomes the container, or parent, of the next object dropped into the interface.

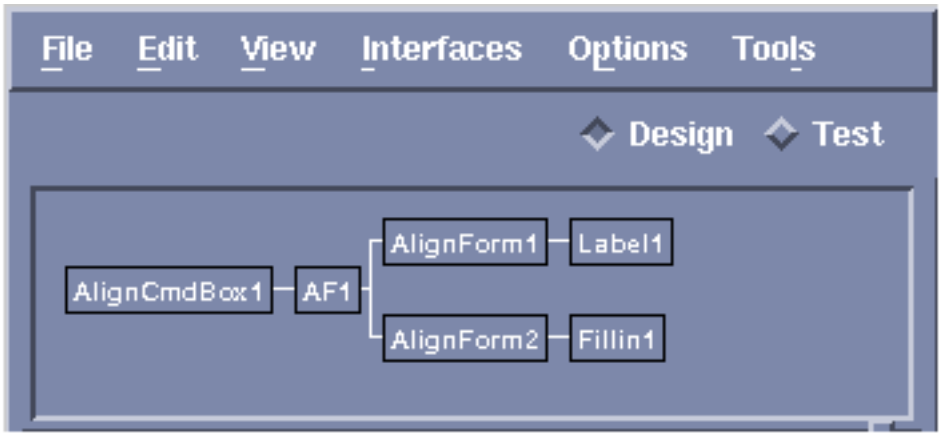
The object selected, such as Fillin1 in the following figure, is surrounded by a box with handles.

Figure 2-4 The Command Argument Box with Fillin1 as the Object Selected

Tree of Objects

The following figure represents the tree of objects in a CAB. The object called AlignCmdBox1, the parent, contains five other objects: AF1, AlignForm1, AlignForm2, Label1 and Fillin1.

Figure 2-5 Tree of Objects



AF1 is the aligning form that was created when the aligning command box was created. AF1, as a parent, contains two objects named AlignForm1 and AlignForm2, and two children, which were dropped into the interface after AF1 was selected.

Fillin1 is a fillin field which was dropped into the interface, as a child to AlignForm2.

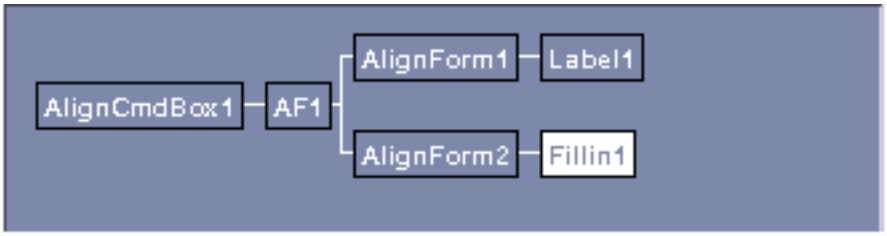
To reparent the fillin field so that it is also a child to AlignForm1, together with Label1:

1. Select Fillin1 with the middle mouse button to reparent.

   The cursor changes to a cross ( ).

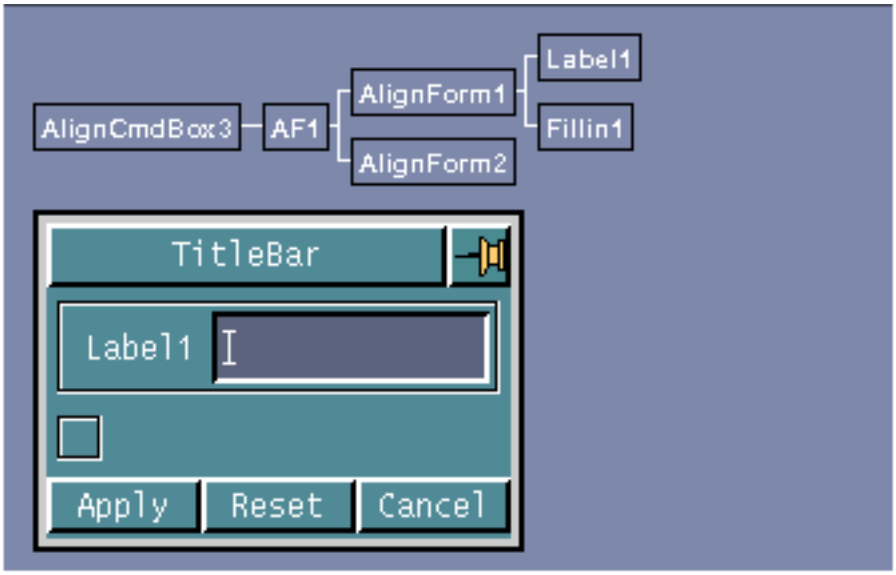
2. Move the cross and its accompanying shadow box over the new parent, in this case AF1.

Figure 2-6 Reparenting Filllin1 with Label1 as a Child to AF1



3. Release the mouse button. The object is associated with the parent as shown here.

Figure 2-7 Label1 and FillIn1 Associated with AF1



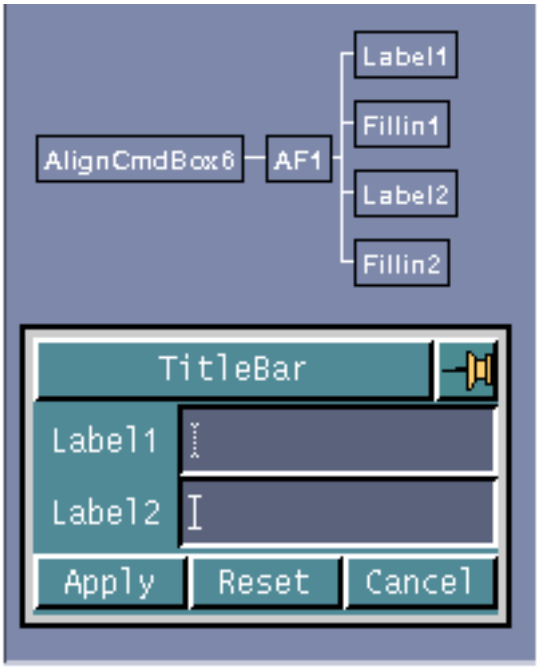
The aligning form itself controls how the children are arranged.

Using the Object Property Editor to Rearrange Objects

You can also specify the rows and columns of a container object. To do so, follow these steps:

1. Select the container whose rows and columns you want to change, for example, the aligning form (AF1) in the following figure:

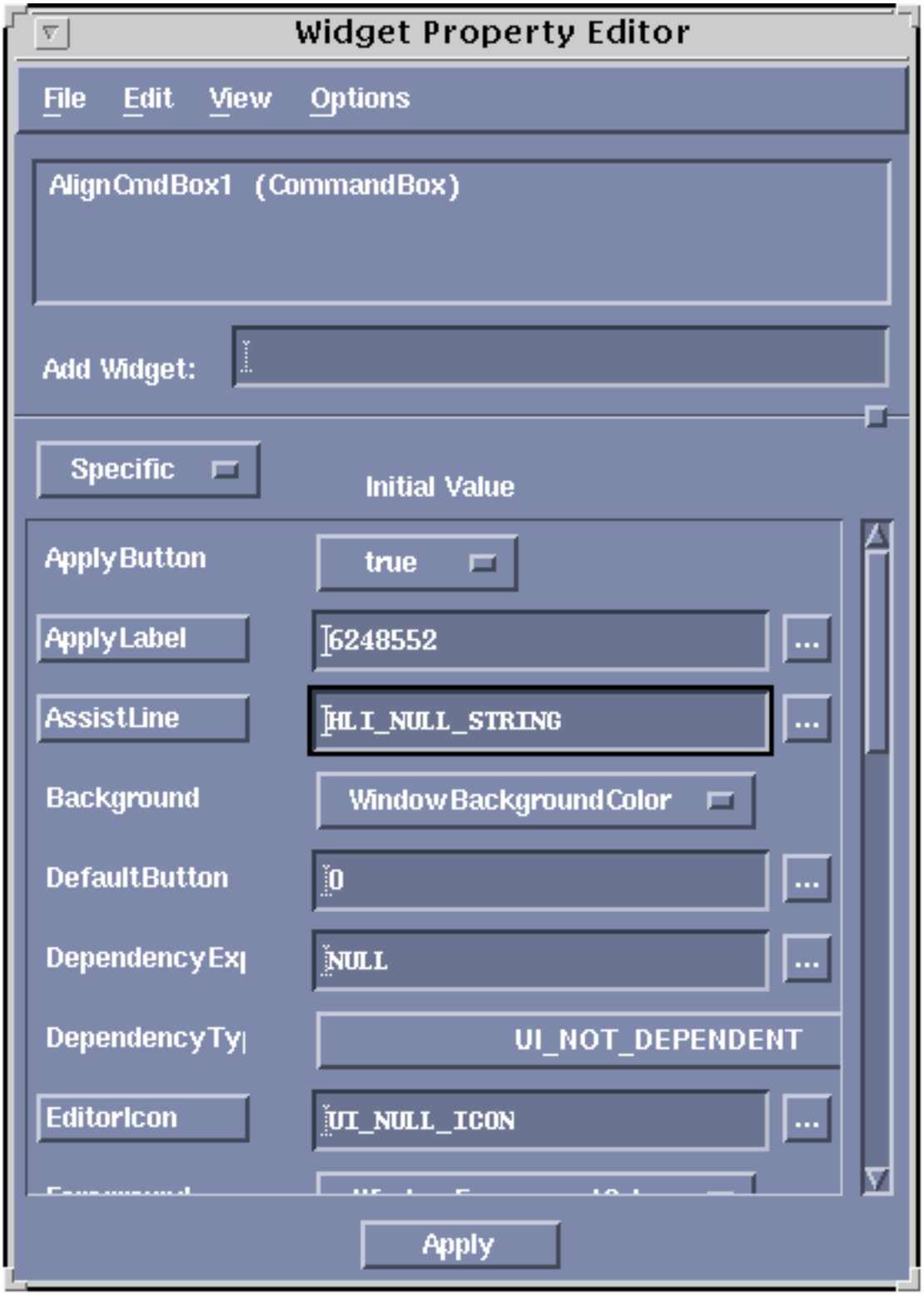
Figure 2-8 AF1Selected to Change Rows and Columns



2. Choose Edit > Property Editor from the menu bar or by using the right mouse button:

   The following Object Property Editor appears.

Figure 2-9 Object Property Editor



3. You can view and modify the properties in different categories from the Property Editor. To change between categories and modify rows and columns, click the button in the box labeled Constraint

   This presents a list of constraint properties which can be changed.

4. To change a property, scroll through the list to select the property to be edited, NumColumns in the category Specific. You can change the number of columns from the default two to four by entering the value 4 in the associated field.

5. Click Apply

   Your CAB now has four columns, as shown in the following example:

Figure 2-10 The Command Argument Box with Four Columns

Changing the Object to be Edited in the Property Editor

To edit the properties of an object other than the one originally selected, drag the newly selected object from either the object hierarchy area of the Advanced Customization Tool or from the dialog box itself. Use the middle mouse button to change the cursor to a cross, and drop the object into the top panel of the Object Property Editor.

Selecting Multiple Objects

To change a common property among many objects, select multiple objects by either of the following methods:

- Holding down the left button and drawing a bounding box or rubber-banding around the ones that you want.
- Selecting the first object, then control clicking the additional ones.
- Selecting all the objects, then control clicking the ones you want to exclude.

  Control clicking involves selection of the first object by pressing the left mouse button, then identifying additional objects by pressing the control key on the keyboard while simultaneously selecting with the left mouse button.

Editing Object Properties

In addition to rearranging objects within a CAB, you can also edit object properties or attributes using the Object Property Editor.

Setting Attributes in the Object Property Editor

The Object Property Editor uses several types of input mechanisms, as described here, depending on the type of data being requested.

Please note: When setting attributes using any of the special editors, you must click Apply or OK while in the editor and then click Apply from the Property Editor to apply the changes.

The Message and Callback Editors

The attribute name may be a raised pushbutton along with a text editor button in the Property Editor as shown below:

Figure 2-11 The Raised Pushbutton is the Attribute Name AssistLine.

When both of these occur, you must use the raised pushbutton. For example, if you click the attribute and it is a

- String, then the Message Editor is called
- Callback, then the Callback Editor is called

The following illustrations show the Message Editor. The Callback Editor is described elsewhere in this book.

If you enter a C expression in the Text Editor window, it is evaluated at run time, and the result appears as the text value for the attribute.

Figure 2-12 Message Editor

If you click on the button showing C:, you can choose the HliString option, for which you can enter text between the double quotes, as shown below.

Figure 2-13 HliString Option

The Text Editor

If the attribute name is not contained within a raised push button, and if there is inadequate space in the fillin field to type all the required information, select the editor button shown to the right of the fillin.

Figure 2-14 Editor Button

The Text Editor appears, as shown here.

Figure 2-15 Text Editor

Enter the required text and click Apply. Next click Apply in the Property Editor window to apply the changes.

Using Option Lists

Attributes with a discrete number of options have an option list instead of a fill in to the right of their names. To select one of the options, click the rectangle in the option list. An option list appears.

Select the desired value. If you select C: ... the Text Editor appears and you can specify a C expression.

Initializing a CVact menu with an option list that numbers more than 10 option items takes a long time depending on the hardware used. For each new option item that you add on the option list, a corresponding Motif PushButton widget is created.

Use option lists for items or options that number 10 or fewer. If you create an option list with more than 10 option items, you cannot see all the option items when you click in the option list because you cannot scroll the list.

Use a scroll list instead of an option list for lists with more than 10 option items.

Recognizing What Has Changed

You may change the values of several attributes before clicking Apply from the Property Editor. To see only those attributes, choose View > Hide Unchanged. For each value that you have changed, CVact displays a vertical bar, as shown below.

Figure 2-16 Changed Value

Click Apply. CVact puts an X to the left of the value if the specified value is incorrect, as shown below:

Figure 2-17 Incorrect Value

Editing Properties of Multiple Objects

To edit the properties of more than one object at a time, select multiple objects. The Property Editor displays the attributes that each of the selected attributes have in common, and it displays the names of the objects in its top panel, with a scrollbar if necessary.

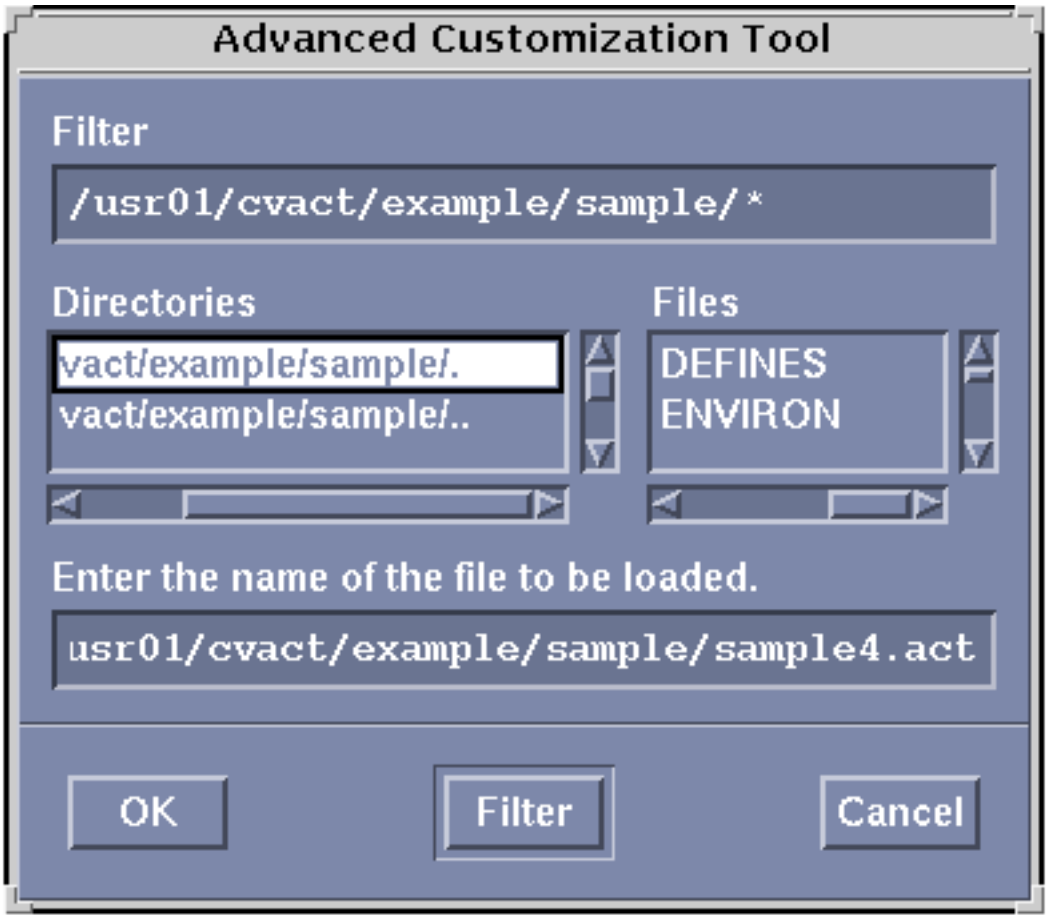
Editing multiple objects is most effective when the objects have properties in common.

Saving the Interfaces

Saving Your Dialog Box

To save the interface, choose File > Save Interface As from the Advanced Customization Tool's File pulldown. CVact adds the CAB name to your working directory as a default name. The system automatically appends `.i` to the name. You must use the cursor to backup and replace `.i` with `.act`. Click OK to accept.

Figure 2-18 The Advanced Customization Tool



Please note: Preferred extension for CVact interfaces is "`.act`" instead of "`.i`".

In this example, after filing, all the information is stored in a text file called `INSCIR.act`.

Using Your Interfaces Using CVact

You can load interfaces into the design tool using the File pulldown.

1. Choose File > Open.
2. Use the same techniques listed in the previous example to supply a file name.

You can also load multiple interfaces into the design tool.

Removing Interfaces

Within the active session you may remove one of the working interfaces from the CVact tool, but without deleting it from the disk by following these steps:

1. Save all changes.
2. Highlight the top level object, such as the CAB.
3. Choose Edit > Delete.
4. Confirm your decision in the popup window.

Chapter 3

Changing Your Interface

This chapter describes how to change layouts once you have created a dialog box. This involves the concepts of dependency, management, and visibility, which are properties that can be set for and between objects using the Object Property Editor.

You can specify columns and rows for placement, and arrange for text to be translated or modified without requiring any change to the layout. You can use icons for push buttons, toggles and labels rather than text, and modify objects.

- Hierarchies
- Managing Geometry and Layout
- Setting Dependency Attributes
- Specifying Layout Options
- Specifying and Using Icons
- Icon Editor
- Icon Converter
- Callbacks
- Using Simple Callbacks
- Customizing Objects

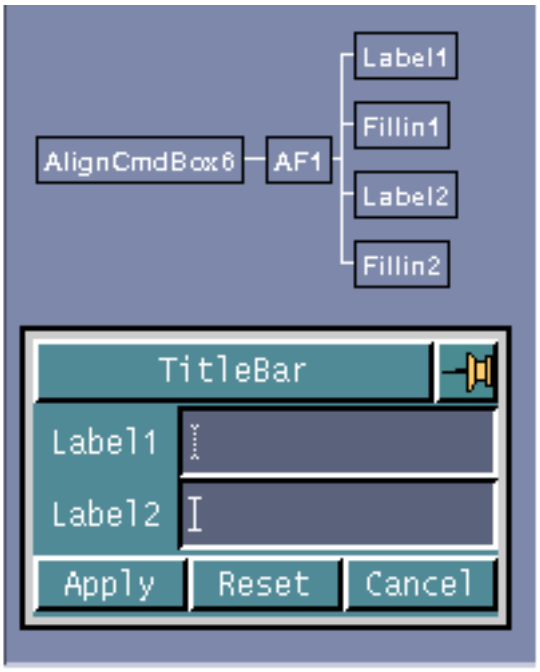
Hierarchies

Hierarchies are a characteristic part of object-oriented software technology.

Parenting Hierarchy

Each object in an interface is part of an hierarchy known as parenting, or instance hierarchy. This means that most objects have parents and/or children. An object that has children simply contains other objects. Thus, in the illustration here, the two labels and fill in fields are children of their parent AF1, and AF1 is a child of its parent AlignCmdBox1.

Figure 3-1 Parents AlignCmdBox and AF1 with Children as Part of Parenting Hierarchy



Instancing Hierarchy

The instancing hierarchy is important in understanding how CVact lays out dialog boxes when you add or change the objects in them, as explained in the next section.

Managing Geometry and Layout

If you use CVact to automatically place and manage the position of objects in your dialog boxes, the effects of translating objects or modifying text on labels, toggles and push buttons is minimized. For the most part, the relative position of the objects is maintained. This section describes the basics behind CVact's automatic positioning of objects.

Geometry Management

The position and size of an object are determined by the attributes or properties of its parent. These rules are the basis of the geometry management. Geometry management considers the child's properties and the parent's available space; this allows the interface to adjust automatically for variations in size introduced by localization, customization, user window resizing, and dynamic changes to displayed data.

Geometry management is performed dynamically at runtime and whenever situations change, for example, if a new field is added or the contents of a field change size.

A field that participates in geometry management with its parent is said to be managed. Every object has an attribute called Managed that can be set. Typically, and by default, fields are managed, otherwise the parent does not allocate space and the field is not visible on the screen. The field below is managed.

Figure 3-2 Field that is Managed

Please note: You would rarely designate a field unmanaged, but you might if you wanted to display a field conditionally, based on values in other fields, in order to keep a command box as small as possible but allow it to grow when the field is to be displayed.

The fillin field in the following figure is not managed. Notice that the label takes up the entire row, and you cannot tell that a fillin exists, unless you look at the parenting tree.

Figure 3-3 A Field that is Not Managed

When a parent decides how to arrange its children, it typically checks each child to compute its minimum and maximum preferred size. Usually, this computation is performed first by checking for specified values; if there are none, the computation is performed according to the child's data content and other attribute values set by the application.

For example, in the absence of specifications, a label computes its preferred width and height based on the text it contains. It considers the font category attribute and the actual font currently in use. Its maximum and minimum preferred sizes are identical, because it always wishes to display its entire contents.

A text fill in field, however, considers its attributes for the minimum and maximum visible length (in characters), and then the font category and actual font in use. If all the characters do not fit, the field can provide horizontal scrolling.

The parent allocates space for each child, depending upon available space. For example, when the user resizes a dialog box, it might be too small for each child's request. In such cases, clipping is automatically performed. If the parent has extra space, the child can specify whether it wants to be centered or justified (top, bottom, left, right) in the space, or whether it wants to be allocated the whole available space.

Visibility

Every object has the property of visibility, which means that it is conceptually drawn on the screen, although it might:

- Be partially or completely obscured by other items
- Exist as an icon
- Lie partially or completely off the actual dimension of the screen
- Have no contents and therefore no pixels to light

If you set the object's VisibleWhenManaged attribute to TRUE, the object is capable of being visible; if it is also Managed, it will be visible when its parent is. The fillin field in the figure below has its VisibleWhenManaged attribute set to FALSE and its Managed attribute is TRUE, therefore it is not visible even though it is managed.

Figure 3-4 A Field that is Managed but Not Visible

The fill in field in the figure below has both its VisibleWhenManaged and its Managed attributes set to TRUE, therefore it is visible.

Figure 3-5 A Field that is Managed and Visible

You can set the visibility attribute to make objects such as command boxes come and go. Most objects are created invisible and only become visible when their VisibleWhenManaged attribute is set to TRUE.

Sensitivity

Every object has the property of sensitivity, which means that if it is set to not sensitive, then it is grayed out or hazed: it cannot receive user input. Both the fill in field and the label in the following figure have had their Sensitive attribute set to FALSE.

Figure 3-6 The Field with the Sensitive Attribute Not Set

The attribute Sensitive means that the object wishes to be sensitive when its parent is. The attribute SensitiveNow can tell you if the object is currently sensitive.

Sensitivity is independent of visibility, so an object can be SensitiveNow even when it is not VisibleNow. Both sensitivity and visibility attributes are subject to special rules for end user customization.

Dependencies

It is common for certain fields to appear only conditionally, depending on values that you supply in other fields.

For example, assume a radio set that offers a choice between English and metric measurements. If you choose English, then an option menu might be tailored to ask for units in inches, feet, and yards. Whether English or metrics units is visible depends on the choice of measurement.

Each field has an attribute to designate that it is dependent. This means that the field is managed, visible, or sensitive only conditionally, depending on the values contained in other fields and expressed in a dependency expression (see below).

This attribute is a DependencyType, and it can have one of three values, each determined dynamically based on the dependency expression:

- Managed affects the value of the Managed attribute
- Visible affects the value of the VisibleWhenManaged attribute
- Sensitive affects the value of the Sensitive attribute

Only one of these three attributes can be under dependency control. The default value for DependencyType is not dependent, which means that the field is not dependent on other fields. In this case, the Managed, VisibleWhenManaged, and Sensitive attributes are only set explicitly by the application.

Dependency Expressions

If an object is dependent, you need to supply an expression to be evaluated at runtime. This is a Boolean expression that refers to values contained in other objects. It can be either:

- A simple expression, for example, the value of a fill in field is non-null
- An expression with other objects as operands whose values are tested by the operators AND, OR, and NOT.

The unary NOT has precedence over AND and OR. The operands are the names of fields in the same dialog box (or names of ApplicationStates). Some examples of operands and their truth values are:

- For a toggle, TRUE means that the toggle is chosen
- For a fill in field or paragraph, TRUE means that it has a non-null string value and is not defective
- For scroll lists, TRUE means that at least one value is selected; if the name doesn't exist, then the operand is FALSE.

A null dependency expression evaluates to TRUE. If the expression is syntactically incorrect in any way, it evaluates to FALSE. The expression is automatically re-evaluated whenever the value changes in any field mentioned in the expression.

As an example, assume that there is a field in a dialog box, for which DependencyType `UI_SENSITIVE`, and the dependency expression attribute is `"englishUnits AND inches"` as it might have been set in the Property Editor as shown in the figure below. Note that the expression must be within double quotes (").

Figure 3-7 Field with Dependency Expression and Dependency Type Attributes Set

Also assume that englishUnits and inches are the names of radio entries in different radio sets. Then this expression means that this field is to have Senstive = true if both the englishUnits and inches radio buttons are chosen; otherwise, this field's Sensitive is false. Whenever the user picks entries in the relevant radio sets, the dependency expression is re-evaluated as necessary, to ensure that Sensitive always has the correct value.

Setting Dependency Attributes

By following the steps identified, you can examine dependencies set up in the sample interface:

1. Change to the directory containing the sample interface by entering the following command:

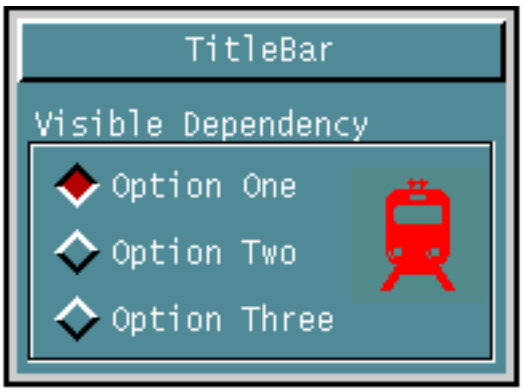
   ```
   # cd /usr/apl/cvact/examples/sample
   ```

2. Enter the CVact tool:

   ```
   # cvact
   ```

3. Select File = Open and choose sample.act. Toggle the Test mode on and select the dependencies.

Figure 3-8 Visible Dependency



4. Change the mode from Test to Design.
5. Use the Object Property Editor to examine the properties of the visible icon that represents Option One. The name of the object should be `Label8`.
6. Verify that the Dependency Expression for the icon reads as follows:

```
"vis1"
```

   This icon has a dependency on an object named `vis1`. As the Dependency Type is `UI_VISIBLE_DEPENDENT`, the icon will be visible when the toggle button is selected.

7. Use the Object Property Editor to verify that the toggle labeled Option One is indeed named `vis1`:

   Click Option One and drag it to the Object Property Editor's object window.

You can put the Sample Main Menu back into Test mode and verify that a different icon appears in the Visible Dependency box, depending on whether you choose Option One, Two, or Three. Because the Dependency Type specifies visibility, and because the toggle buttons are in a radio set and only one can be TRUE at a time, only one of these icons can appear at a time.

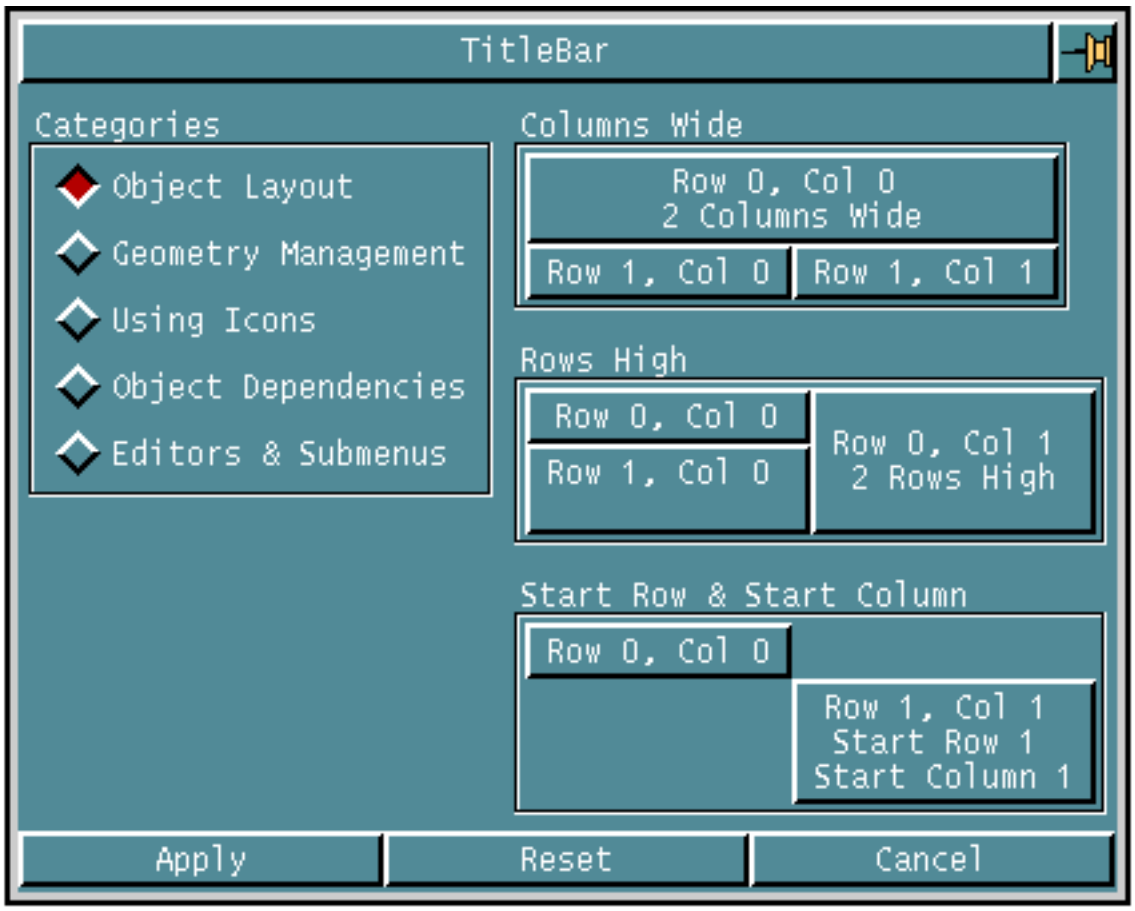
If the Dependency Type specified sensitivity, as it does for the radio set labeled "Sensitive Dependency," then the object possessing the dependency would become not sensitive or hazed as its toggle is selected off by clicking another toggle of the radio set.

Specifying Layout Options

Because CVact does layout automatically, there are times when you may wish to specify a different display for an object, such as the row and column position at which it is displayed and the number of rows and/or columns in which it is displayed.

You can select Geometry Management toggle in the Sample Main Menu to display the basic options that result when you specify various positional and size attributes, as seen in the following figure.

Figure 3-9 Sample Menu with Basic Options



Specifying and Using Icons

You can assign icons to many of the objects in CVact. These icons must be in XPM (X Pixmap) format. You can

- Make your own icon, using `iconeditor` (the icon editor command)
- Convert icons that you have used previously in CADDS (using the `iconconv` command)

Please note: Find Pixmaps elsewhere and use them.The Icon Editor works only on Solaris. You can use any icon editor that supports XPM format.

Using the Object Property Editor, click the Icon button to launch the Pixmap Browser.

Use the filter popup box to select the directory and the particular icon. You can think of this as an icon browser because it displays the icon as you select a Pixmap file.

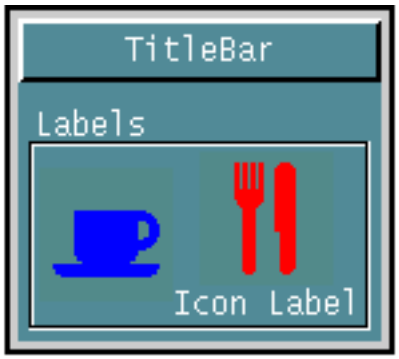
Figure 3-10 Pixmap Browser

Icon Examples

In the Sample Main menu you can see various icons, some with labels, some without, and some that depend on the state of a toggle.

The following figure displays a cup and saucer, without an icon label, and a knife and fork, with an icon label. If you use the Object Property Editor to look at their properties, you will see that the TextValue attribute for the cup and saucer is set to `HLI_NULL_STRING`. If you specify a TextValue, that specification will become the icon's label automatically.

Figure 3-11 A Menu with Icons With and Without Labels



The icon for the knife and fork has a TextValue property value.

Icon Editor

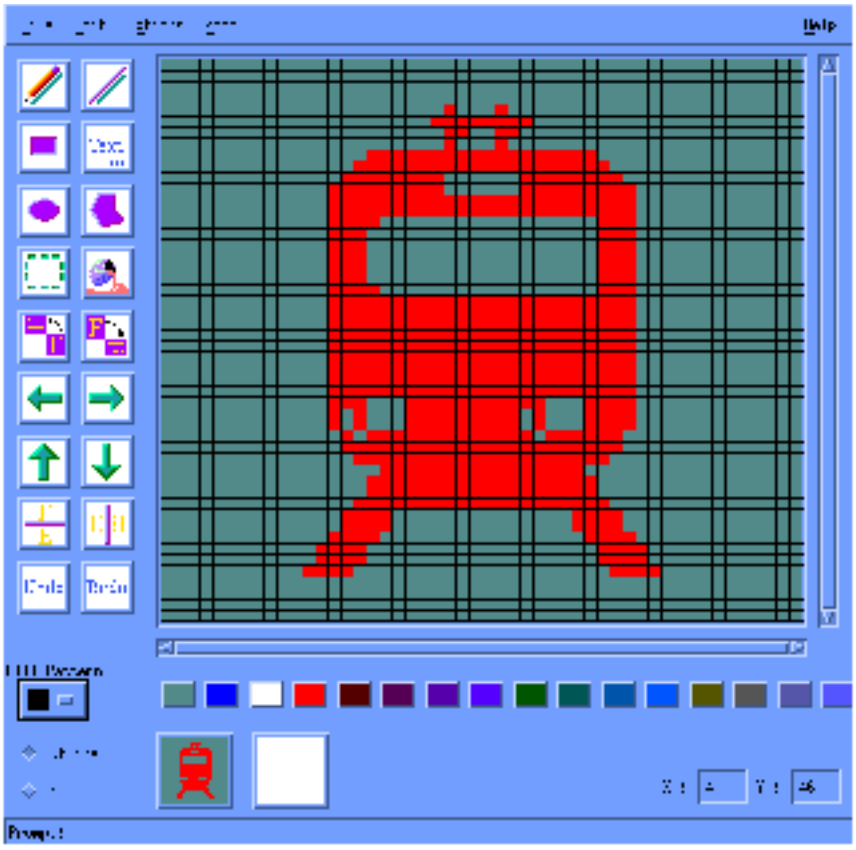
The icon editor provided with CVact uses icons that it finds in the `/usr/apl/cvact/data/reposit/C/iconed.d` directory. Without a pointer to these icons, the editor will not be able to display its own icons for the push buttons. To set this up, use the command:

```
# setenv ICON_EDITOR_ICONS
   /usr/apl/cvact/data/reposit/C/iconed.d
```

The editor itself can be run using the command:

```
# /usr/apl/cvact/bin/iconeditor     or
# iconeditor
```

Figure 3-12 Icon Editor



The icon editor works with pixels and `xpm`, the X Pixmap icon format developed for including images in X programs. Previously, the CADDS icon editor used vector based images rather than pixels, and a proprietary format rather than `xpm`.

The editor has most of the usual properties of an icon editor. You might want to take these steps to familiarize yourself with it:

- Try moving the cursor over the buttons on the left hand side of the window. You will notice at the bottom of the window, next to Prompt:, that a short description of each button appears.
- Pick a current color by selecting one of the 16 color buttons located along the bottom of the window.
- Using the tools on the left, attempt to create an icon. Some things to keep in mind:
  - Select Fill at the bottom left of the screen if you want your circles, rectangles, and so on to be flood filled when first drawn.
  - Undo/Redo will continually step backwards/forwards through any steps you take.
  - You can clear the screen under the 'Edit' menu.
- Save your icon using File = Save As and specify a name, such as `myicon.xpm`. (Use the extension `.xpm` to remind yourself that this is in the "xpm" format.)

One of the features of this editor is the camera. (The push button is named Insert Camera Here.) This creates a rectangle whose size matches the grid that you have chosen for your icon. You can move the camera lens around the screen. When you click again, the image under the lens appears in the grid. Another feature is the button named Text... that lets the editor transfer text that you type into a popup window (in your choice of fonts) from the window to the icon grid.

Icon Converter

The icon converter converts from the company's proprietary format, vector-based images, to X Pixmap icons that can be used with CVact. It will convert either a single icon at a time, or an entire library of icons.

Syntax

```
iconconv options name
```

Description

`name` is the name of the icon library

`options` can be one or more of the following;

-d destination_directory_name
-i individual_icon_name
-c menu_color
-t conversion_format
-r desired_resolution
-h new_icon_height
-w new_icon_width

By Default:

- The current working directory is the destination name
- An entire library is converted
- The menu color is `menudarkgreen.mcol`
- The conversion format is XPM 3
- The resolution is 75dpi

Example

The following example puts a single converted icon into the directory `/tmp/composer/piggy.xpm`:

```
# iconconv -d /tmp -i piggy -c menured.mcol -r 100dpi
composer
```

Callbacks

Callbacks are functions that are registered to a particular object. (The CVact IDT automatically registers the callback code that you write, if you use the UI Callback editor.) Callbacks vary from object to object. If the object is a push button it has a callback named PushedCb. When you press that push button, the code that is associated with the PushedCb callback gets executed.

Although callbacks do not have to be associated with a user action, their main purpose is frequently to interact between you and that part of the application that reacts to the specific user event.

Types of Callbacks

This section lists all the available callbacks and describes when each of the callbacks is invoked. See the appendix on Attributes for information on:

- Description of the callback functionality
- The callback's system data pointer

Table 3-1 Callback Descriptions

Callback	This callback is invoked when:
ApplyCb	You have pressed the OK or Apply button.
CancelCb	You have pressed No or Cancel or some similar button.
ChosenCb	(1) A toggle or option list value is set internally by copying from the default or remembered values; (2) you change the state of a toggle; (3) you choose an option list item; or (4) a scroll list item is chosen or not chosen by you or internally from default or remembered value. Not called when a toggle or option list value is explicitly set via UiSet...( ) call. For scroll lists, System Data is a transient pointer to TextValue. It is a transient UiChosenStruct that contains item string, item position, new chosen value: TRUE or FALSE, and modst : modifiers state. `Modst` is a bitmap with each bit representing an X toolkit Intrinisics (Xt) modifier. Only the Keyboard `Shift` key modifier represented by the first right most bit is used. Use the mask UI_SHIFT_MOD_MASK to test the state of the keyboard `Shift` key modifier.
ClosedCb	You close (terminate) an object.

Table 3-1 Callback Descriptions

Callback	This callback is invoked when:
CompleteCb	A field value is loaded from default, or you have: (1) released the mouse button to drop a form that has been dragged—this callback is made after the drop, but before the final painting of the contents is complete; (2) released the slider to complete a slide operation; or (3) moved the focus out of a field, for example, by tabbing to the next field or by trying to confirm the parent dialog box contents. CompleteCb gets called when the field loses keyboard focus. Keyboard focus is different from mouse focus. A field has keyboard focus if it can accept any character entered from the keyboard. A field is said to have lost keyboard focus when it can no longer accept any character entered from the keyboard as in the following instances: o Click in any other Fillin, IntFillin, FloatFillin, or Paragraph field. o Click outside the CVact property sheet. o Press the `Tab` key or press the Shift and Tab keys together (Shift key simultaneously pressed with the Tab key) to move the keyboard focus to the next or previous Fillin, IntFillin, FloatFillin, or Paragraph field. o Click `Apply` on the Command Align Box (CAB) after modifying a field. o Press `Carriage Return` after completing a field.
CompleteCellCb	A cell in a scroll matrix has been edited.
ConfirmCb	You acknowledge the warning, usually by pressing Yes.
DefaultActionCb	An item is invoked for default action, for example, double-clicked. For scroll lists, System Data is a transient pointer to TextValue. It is a transient UiChosenStruct that contains item string, item position, new chosen value: TRUE or FALSE, and modst : modifiers state. `Modst` is a bitmap with each bit representing an X toolkit Intrinisics (Xt) modifier. Only the Keyboard `Shift` key modifier represented by the first right most bit is used. Use the mask UI_SHIFT_MOD_MASK to test the state of the keyboard `Shift` key modifier.
DestroyCb	The object is about to be destroyed. As soon as the callback returns, the object will be destroyed. This callback is not performed during fast shutdown.

Table 3-1 Callback Descriptions

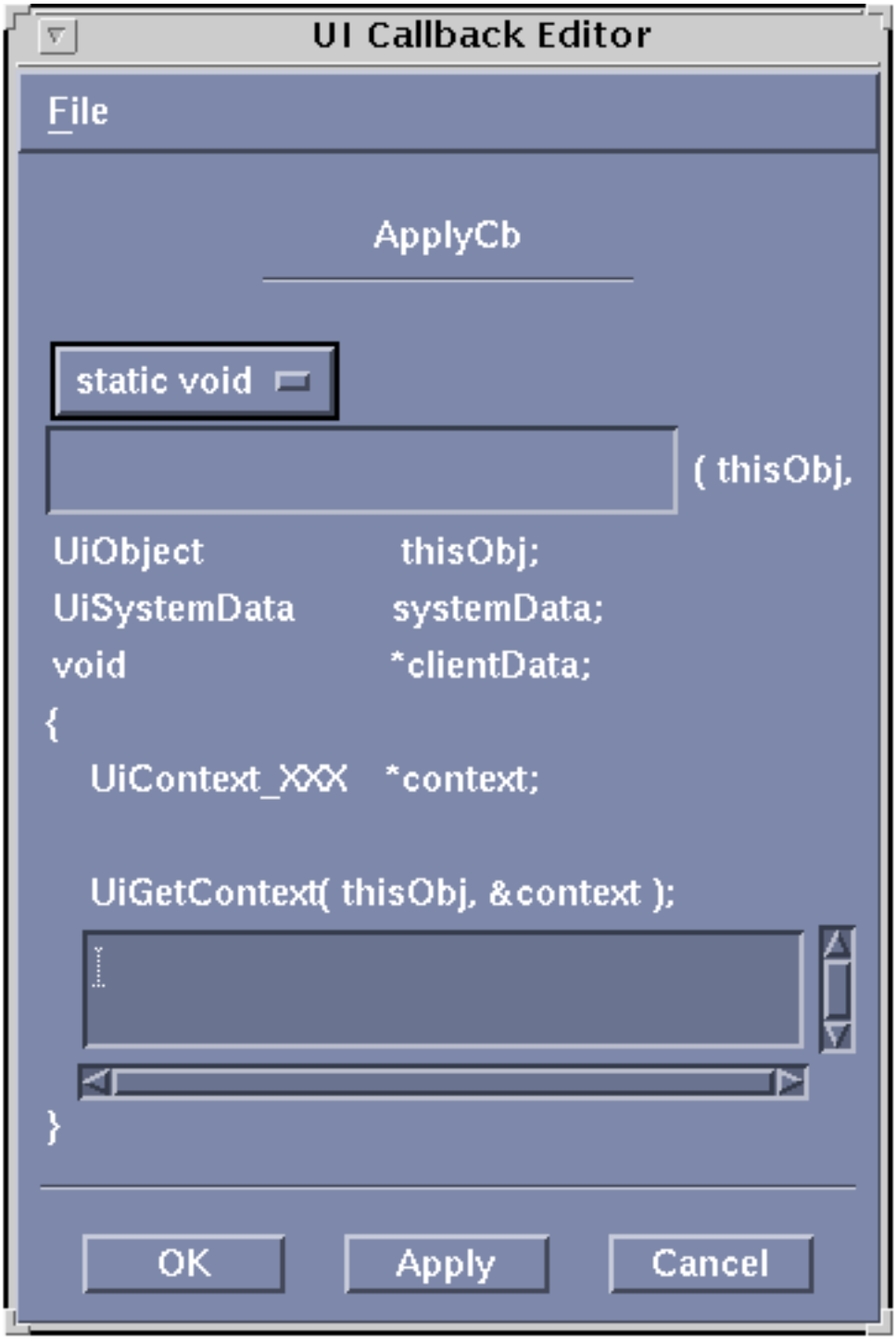
Callback	This callback is invoked when:
DropCb	A drag form is dropped on the field with the callback.
EditorFocus ChangeCb	Focus of attention moves to a fill in field. The new field's associated editor dialog box performs the EditorFocusChangeCb.
EditorInvokedCb	You press the Editor Button.
EditorRescindCb	You close a dialog box containing a fill in field. The Editor dialog box associated with each fill in field invokes that field's EditorRescindCb.
HelpCb	Context-sensitive help is requested. (for example, when UiSetContextHelpMode has been called, and you click on a building block). Also can be called under other circumstances.
InitializeCb	A field is initialized or reset, e.g, each time its parent dialog box becomes visible, but after the field is prefilled from default or remembered values (if appropriate) and before the field is actually displayed. For scroll lists, since Prefilled=Inferred, the InitializeCb is called the first time the Scroll List is about to be displayed.
InputCb	Continuously as you change the value of a slider, or immediately after you change the TextString in an input paragraph (for example, by typing a character, deleting a character, pasting, and so on.). Not called when the application sets the value or string explicitly or from default or remembered values.
IsDropOkCb	A sibling drag form is being dragged and is on top of this field.
IsEditableCb	Immediately before a scroll matrix's cell is to be edited.
IsInputOkCb	Every time you attempt to change the TextString in a field; or text in a scroll matrix, in which case this callback is called after each edit (adding or deleting text) inside a cell.
PushedCb	A button is pushed.
ReportCb	You press the Problem Report button in a Failure box.
ResetCb	You press the Reset button.
SelectCellCb	You select a scroll matrix cell.
StartCb	You begin a dragging operation.
Window ActivateCb	Focus of attention moves from one dataset window to another.

Using Simple Callbacks

Arguments Passed to the Callback

When you write a callback using the UI Callback Editor, the wrapper for the callback is in place for you, so that you only have to write the contents of the callback function. You cannot put in the function declaration or the braces.

Figure 3-13 UI Callback Editor



Callbacks by default are `static`. This means that this callback cannot be called by any function outside of the module (file) which contains it. In this case, the ApplyCb function will be placed within the C++ module which defines all of the

interfaces for the current project. If you change the callback to `extern`, then it can be called by such external functions.

thisObj

The first parameter sent a callback is a pointer to the object that contains it. In the previous example, the apply callback is used for the Apply button in a Command Argument Box. You cannot declare or define the CAB's object pointer, but you may use it if you wish.

systemData

System data is passed to callbacks when they are invoked. Often this data is passed by means of a `transient pointer` that points to data in memory managed by the Ui subsystem. This data may be a copy or actual internal data, thus you do not want to modify it unless this possibility is explicitly stated in the description of the callback attribute.

Such pointers are valid only for the duration of the callback. References to this memory after return from this callback are incorrect and likely to have disastrous consequences.

clientData

You can pass a pointer to your callback that points to some application data. To do this, specify the pointer in the ApplyCbClientData attribute for your object. This pointer will be sent as the clientData argument for the callback.

HLIString Manipulation Utilities

In your application program, you can use C strings `(char *)` as you would normally, for instance for:

- Certain C language format statements
- Character string data for external files
- Defining C language expressions

However, most of the UI functions require what we call `HLI Strings` because these functions are designed for international usage. Therefore, the character type for internal processing is `HliChar`. HliChar is an integer type, usually a word or long. The corresponding string type is `HliString` and is defined as a pointer to an HliChar. There is also a fixed array type `HliShortString`, which is an

HliString with a maximum of 256 characters. An HliChar with the value zero is reserved as an end-of-string marker.

The third type of string is the C string (char and char *). These will sometimes be referred to as CChar and CString. These C character and string types can still be used, but because they cannot be translated for use world-wide, they are inappropriate for most user-visible text. The normal character and string functions are assumed to be provided by the platform.

A Callback determines what happens before or after a specific event occurs.

Callbacks can be left for various types of events. A simple event would be when you press the Apply button in a dialog box. This type of callback is an ApplyCb.

As with text editors, callbacks have their own editor.

Using the UI Callback Editor

The following steps lead you through an exercise in using the Callback editor:

1. Create a simple CAB as shown in the following figure:

Figure 3-14 A Simple Command Argument Box

2. Bring up the Object Property Editor window on the CAB.
3. Under Behavior, select ApplyCb to display the Ui Callback Editor.
4. Type the following C statement into the paragraph field:

```
printf ("You just pushed the Apply button\n");
```

5. Click the Apply or OK button in the Callback editor.
6. Click Apply in the Property editor.
7. Toggle the Test mode.
8. Click the Apply button in the CAB.

9. Look for the following message to show up in the message area of the Advanced Customization Tool:

```
You just pushed the Apply button
```

Creating a Simple Dialog Box

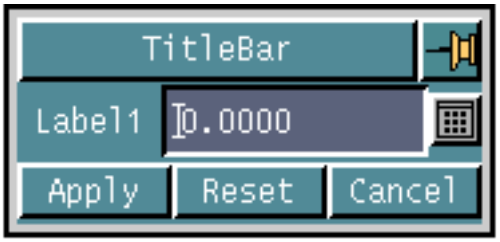
The following directions create a simple dialog box.

1. Create a simple CAB.

Figure 3-15 A Command Argument Box

2. Insert a LABEL and a Floatfill from the Palette: UIObjects window.

Figure 3-16 The Command Argument Box with Label and Floatfill Inserted



3. Bring up the Object Property editor on the CAB.
4. Change the CAB object name to a more applicable name under Declaration and click Apply.
5. Change the property CAB name to INSERT CIRCLE by modifying the WindowTitle located under Specific.

Figure 3-17 Object Property Editor

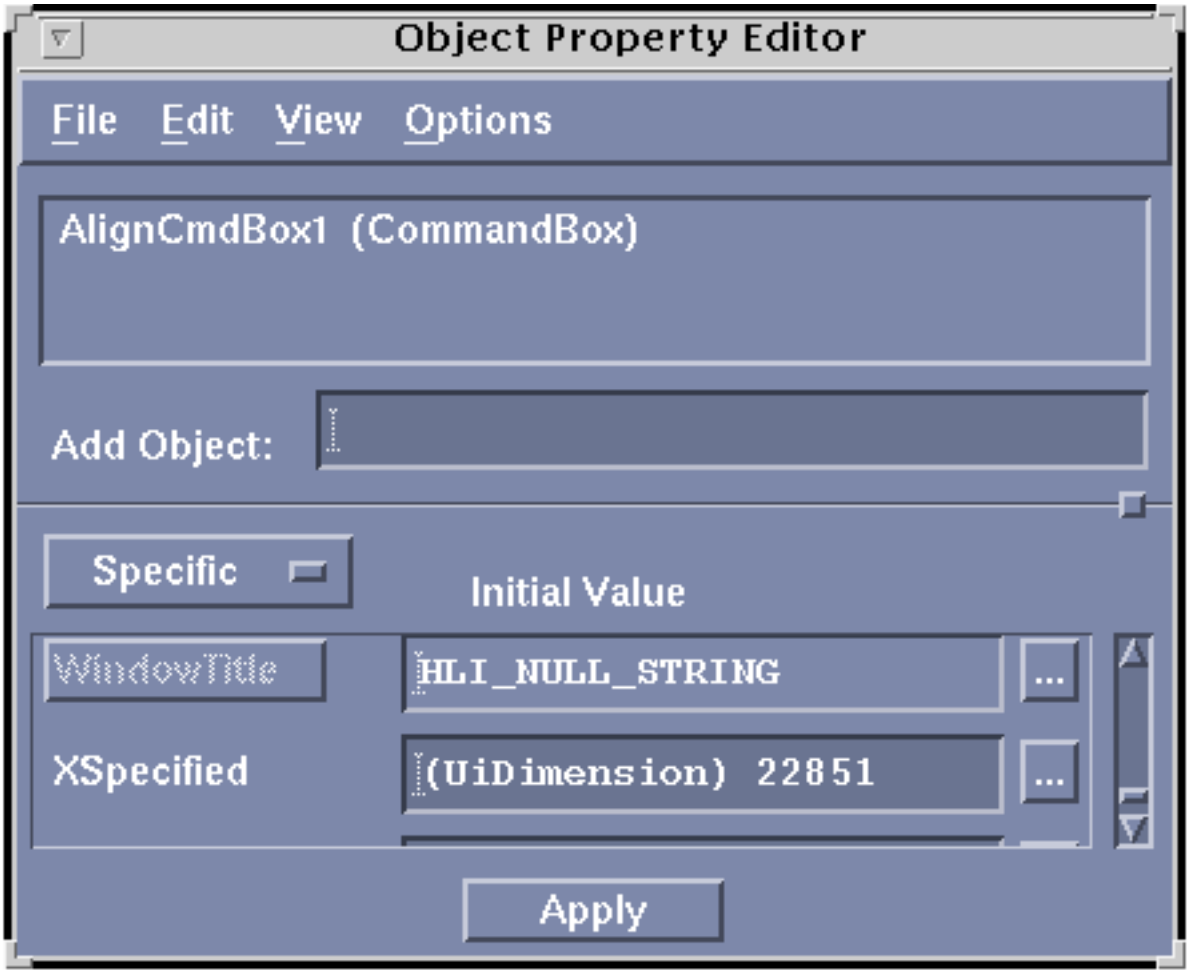
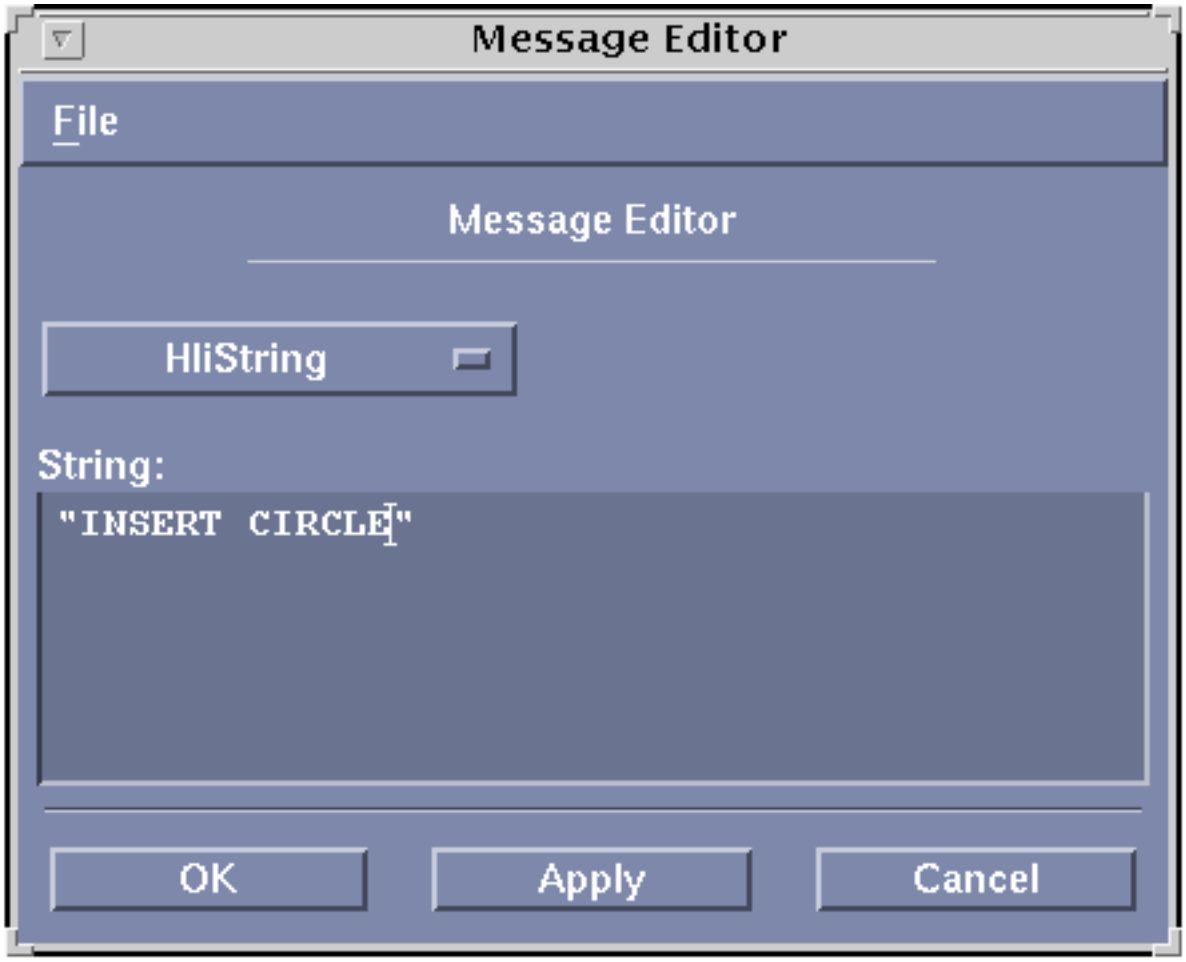


Figure 3-18 Message Editor



6. Click OK in the Message Editor and Apply in the Property Editor.
7. Drag and drop the label into the Object Property Editor.

A number of code fragments have been included to be used in the callbacks. They are intended to perform common functions that you may need to use in your interface. They can be modified to accomplish a specific task, or used as written. The directory containing these fragments is `/usr/apl/cvact/examples/CODE`.

The following is a list of these code fragments. Note that there are no spaces in the names.

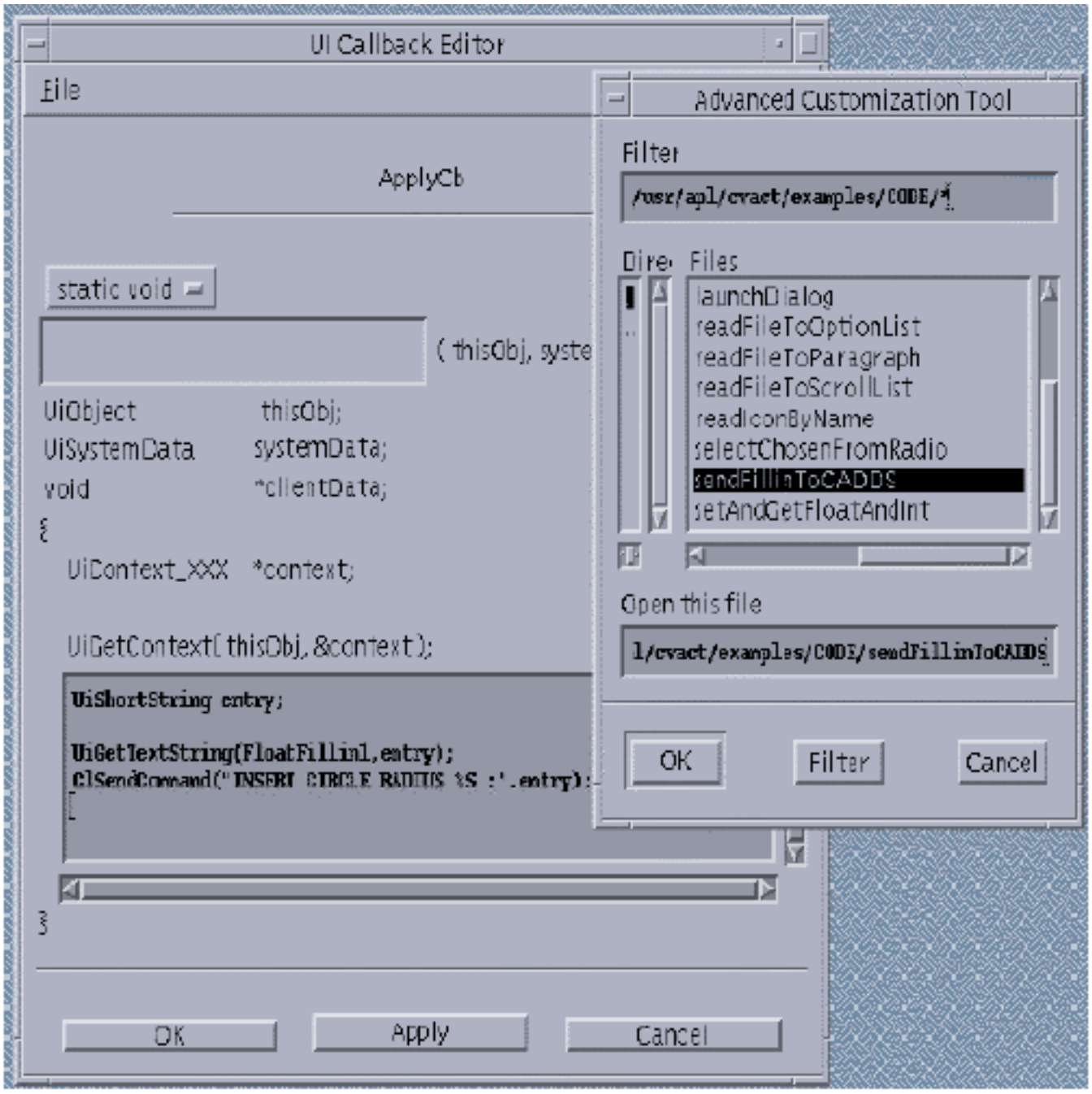
- changeColor
- choseRadioOption
- closeOtherCabs

- fillInOptionList
- getChosenNameFromScrollList
- launchDialog
- readFileToOptionList
- readFileToParagraph
- readFileToScrollList
- readIconByName
- selectChosenFromRadio
- sendFillinToCADDS
- setAndGetFloatAndInt

We can use the code fragment `sendFillinToCADDS` to continue with the previous example. As its name implies, the file contains C code that sends the value of a fill in field to CADDS.

1. To bring up the UI Callback editor, select ApplyCb from under Behavior.

Figure 3-19 The UI Callback Editor



2. Select File => Open.

   This will cause the file selection box to open.

3. Change the directory to `/usr/apl/cvact/examples/CODE`

4. Insert the `sendFillinToCADDS` code fragment into the UI Callback editor (you may want to link or copy this directory to your home directory)

   This code fragment does not require any editing for use in this example. The Float fill in name is `FloatFillin1` and the command `ClSendCCommand`, which behaves like a print statement in C, includes the correct message to be sent to CADDS. The text value of the `FloatFillin1` is read into the variable `entry` and is printed to CADDS through the command `ClSendCCommand`.

5. Click the OK button. When you apply this change to the Property editor, the C program will be automatically compiled for syntactical errors.

6. To test the interface, select the Test toggle from the Advanced Customization Tool window and enter a value, either through the calculator or the keyboard, for the circle.

7. Click Apply. The message is printed in the message area of the window.

   The message `Send Command to cadds:` appears in order for you to differentiate between `printf` and `ClSendCCommand`. It will not be sent to CADDS

8. File the interface under the name `INSCIR.act`.

Customizing Objects

Customizing Most Attributes

Many attributes can be customized from values provided in resource files. Resources are accumulated from a variety of sources, including system-wide, project-wide, and personal resources files.

Resources

The resource combines the name of the object and the name of the attribute, for example:

```
createFillet*radius=0.5
```

This means that in the dialog box named createFillet, the field named radius will have the value 0.5 for the Default attribute.

Attribute customization resources are processed when the UiObject is created (in this case, when the field named radius in the dialog box named createFillet is created). At this time, the value specified in the resource overrules the value provided in the input argument to the creation method.

Another way of saying this is that, for customizable attributes, the values provided by CVact when an object is created are only suggested initial attributes for the UiObject, to be used in case there is no customizing resource. If your program invokes a `UiGet...()` immediately after creation, the value gotten may not be the same as the one that was provided in the `UiCreate...()` method.

Since customization resources are only read at the time the UiObject is created, your application should not set these resources after object creation, because the intent of the user may not be satisfied and the user will likely perceive this as a bug.

Therefore we suggest that you set customizable resources only at object creation time. Most customizable resources are unlikely to be needed to change at runtime.

Sensitivity and Visibility

Special rules pertain to two attributes: Sensitive and VisibleWhenManaged. There are default behaviors for these attributes, which can be overridden in the callbacks.

Therefore, customizations of these two resources are handled in a special way. Sensitivity and visibility are each internally represented by two underlying values: one which is under application control, and one which is under user-customization control (and unavailable to the application). The object is actually visible on the screen only if both the application-set value and the user-set value are TRUE (and similarly for sensitivity). User customization of sensitivity and visibility are therefore handled automatically, and the application can freely set or get the visibility or sensitivity attributes without concern for user customization.

The effect is that if these attributes are customized to FALSE, then the object is insensitive and/or invisible permanently, regardless of how the attribute was set by the application. Also, dependency expressions that might make the object sensitive or visible do not actually cause the object to be sensitive or visible on the screen. However, if these attributes are customized to TRUE or left uncustomized, then the object is insensitive and/or invisible.

Color and Font Category

Colors and fonts are defined in resource files so that you only have access to the options available on the option lists for the following attributes:

- Foreground color
- Background color
- Font Category

In order to allow user preference customization of all the fonts and colors in an application, the application does not directly specify fonts or colors as attributes of objects. Instead, UiObjects of class Color or FontCategory are referenced as attributes of BuildingBlock objects.

You can customize the particular font set used by a FontCategory and/or the particular screen color associated with a Color object.

An object of either the Color or FontCategory classes contains attributes that specify the desired appearance (color or font characteristics). When the object is created, attribute values are set, but you can override them with attribute customization.

A number of generally-useful colors and font categories are predefined, and meaningful (platform-dependent) default colors and font sets are associated with them. Additionally, the application UI developer can define new colors and font categories when appropriate, using the appropriate `UiCreate...()` method. For example, an application with a special calculator might use a symbol font for some

of the calculator buttons; these would be represented by a calculator button font category.

Every BuildingBlock has attributes which specify the foreground and background colors. Normally, the default values are used (the defaults differ depending on the class of BuildingBlock). For example, dialog boxes are drawn with their background in the color specified by `UI_APP_BACKGROUND_COLOR`. But if the application desires, it can set these attributes to be other colors, either predefined or application-created.

Similarly, BuildingBlock classes that display text have attributes that specify the font category to be used. BuildingBlocks, for example, ScrollMatrix, which display many pieces of text for differing purposes may have more than one attribute specifying the appropriate font category for each. These attributes can be left at their default (which depends on the class), or can be explicitly set to other font categories, whether predefined or application-created.

Following are tables of predefined font categories and colors, listing the object names and the predefined constants of type UiColor and UiFontCategory which refer to each. The customization name is used in resource files for attribute customization.

Table 3-2 Font Resources

Font Resource Names	Default
LabelFont	UI_LABEL_FONT
MajorLabelFont	UI_MAJOR_LABEL_FONT
MenuFont	UI_MENU_FONT
MinimalLabelFont	UI_MINIMAL_LABEL_FONT
MinorLabelFont	UI_MINOR_LABEL_FONT
MinorParagraphFont	UI_MINOR_PARAGRAPH_FONT
MinorTextFieldFont	UI_MINOR_TEXT_FIELD_FONT
ParagraphFont	UI_PARAGRAPH_FONT
PopupFont	UI_POPUP_FONT
PushButtonFont	UI_PUSHBUTTON_FONT
TextFieldFont	UI_TEXT_FIELD_FONT
Title Font	UI_TITLE_FONT

Table 3-3 Color Resources

Color Resource Names	Default
AlertBackgroundColor	UI_ALERT_BACKGROUND_COLOR
AlertForegroundColor	UI_ALERT_FOREGROUND_COLOR
ApplicationBackgroundColor	UI_APP_BACKGROUND_COLOR
ApplicationForegroundColor	UI_APP_FOREGROUND_COLOR
ScrollbarBackgroundColor	UI_SCROLL_BACKGROUND_COLOR
WindowForegroundColor	UI_WINDOW_FOREGROUND_COLOR
WindowBackgroundColor	UI_WINDOW_BACKGROUND_COLOR

The application foreground and background are used for the Advanced Customization Tool window and for most dialog boxes. The window colors are used for dataset windows and presentation areas. The alert categories are used in Failure and Error dialog boxes.

Attributes of Font Categories

Characteristics of a font category can be assigned either by setting attributes for typeface, style, and size, or else by setting an all-encompassing Specification attribute. Setting the Specification implicitly sets the typeface, style, and size, and vice versa, so the attributes are never contradictory.

The Specification is a platform-independent description, represented as a Cstring consisting of typeface, size, and style fields separated by commas. The typeface is a string consisting of alphanumeric characters, or a string of any characters surrounded by double quotes. The size is in points. Supported styles include plain, bold, italic, and bold-italic. Any of the fields can be omitted, with the defaults being system specific. When fields are omitted, trailing commas can be omitted as well.

Examples of valid font Specifier strings include:

- Courier,12
- Courier, 18, bold-italic
- HELVETICA, 24, BOLD
- “Computer ISO 8859-1”, 12, plain
- Italic

Note that this portable form does not allow specification of font sets, that is, multiple fonts each covering different ranges of the character encodings. A single portable font specification may actually be implemented as a font set including several fonts for different ranges of the encoding. Platform-specific font strings

may be supported in some cases. But in the platform-specific case, a particular font does have an encoding, and, therefore, it may be necessary to list several fonts (a font set) in order to have all the characters that an HliString may reference. A font set is a comma-separated list of fonts.

Attributes of Colors

The particular screen color associated with any color object is assigned in the Specifier attribute. The Specifier attribute may be set explicitly via `UiSet...()`; also, convenience routines are available to set or get the Specifier attribute using Red/Green/Blue (RGB) or Hue/Lightness/Saturation (HLS) values.

The format of a Specifier allows either an RGB triple or a “Crayola” color name. Syntax for the RGB triple form is red, green, blue, where the red, green, and blue values are floating point values ranging from 0.0 to 1.0 and representing the intensity of the respective color component.

The Crayola colors are specified by name. Case is ignored.

Table 3-4 Crayon Color Names

Color	Color	Color	Color
apricot	fuchsia	olive green	sepia
aquamarine	gold	orange	silver
bittersweet	goldenrod	orange red	sky blue
black	green	orange yellow	spring green
blue	green blue	orchid	strawberry
blue green	green yellow	peach	tan
blue grey	grey	periwinkle	tangerine
blue violet	indian red	pine green	teal
brick red	jungle green	pink	thistle
brown	lavender	plum	turquoise blue
burnt orange	lemon yellow	purple	violet blue
burnt sienna	magenta	raw sienna	violet red
cadet blue	mahogany	raw umber	white
cerulean	maize	red	yellow
copper	maroon	red orange	yellow green
cornflower	melon	red violet	yellow orange
cyan	midnight blue	royal purple	
dandelion	mulberry	salmon	
forest green	navy blue	sea green	

Platform-specific color strings may be supported in some cases.

The syntax for the system default colors is `@name` where the name is used to find a color as specified by the system. This facility is included to allow the use of Windows system colors, which the user controls through the control panel. Names to be supplied. Note that this is platform specific.

The application foreground and background are used for the main menu and for most dialog boxes. The window colors are used for dataset windows and presentation areas. The alert categories are used in Failure and Error dialog boxes.

Examples of valid color specifications include:

- 1.0,0.0,0.0
- indian red
- @WindowBackground

Chapter 4

Executing Your Menus

When you finish designing your interface, you can use this chapter to generate code, compile, load and create either shared libraries to be used by CADDS or your own application.

- Generating Shared Libraries
- Using CVact Menus with CADDS on Windows
- Using the Sample Menu
- Customizing Your Menus

Generating Shared Libraries

The following is the content of the file:

`/usr/apl/cvact/Install/HowToLoadAndLinkToCADDS`

Creating Shared Libraries

Create a shared library, `libUserCr.so` for UNIX machines and `libUserCr.dll` for Windows, needed to run your new menus in CADDS in conjunction with a CR process following the steps outlined. You can use your `CVUI` executable as a quick test of your CADDS menus or for your standalone menus. You cannot use the `CVUI` executable on IBM and Windows.

1. Go to the directory where your interface (`.act` file) is located by entering the following commands:

   ```
   # source ~/.cvactrc
   # cd
   # cd example/inscir
   # ls InsCir.act
   ```

2. Copy the `.act` files created on UNIX to `/usr/apl/cadds/lib/UserCR/` on the drive where CADDS is installed on Windows.

3. Enter `share_cvact` at the UNIX shell prompt: as follows:

   ```
   # share_cvact
   ```

   On Windows, enter `share_cvact` in the korn shell supplied by NutCracker as follows:

   ```
   perl /usr/apl/cvact/tools/NTscripts/share_cvact.pl
   ```

   At this point, the system creates C++ code from your `.act` file. It also creates supporting files. These files are compiled and linked into a shared library `libUserCr`.

   Listing the directory returns the following (on Solaris):

   ```
   Dependfile   app.h        appinitU.o    InsCir.act
   Makefile     appinitU.cxx libUserCr.so  InsCir.o
   ```

   There is an object file (`.o`) for every menu file (`.act`) file.

   On Windows, it creates the `libcvUserCR.dll` in the `/usr/apl/cadds/bin` directory.

Standalone Testing Using CVUI (supplied under /usr/apl/cvact/bin)

The `CVUI` executable can be used as a quick test for user created menus. This executable is not available on Windows.

You are not required to enter CADDS to use `CVUI`.

1. If the `libUserCr.so` file is not in your current directory, you must modify the `LD_LIBRARY_PATH` (on Solaris) to point to it.
2. Verify that you are using `/usr/apl/cvact/bin/CVUI` by entering the following command at the UNIX shell prompt:

   ```
   # which CVUI
   ```

3. List the libraries to verify that they are all defined and pointing to either `/usr/apl/cvact/lib` or the UNIX libraries by entering the following command at the UNIX shell prompt:

   ```
   # cv_ldd `which CVUI`
   ```

   If they are undefined and you cannot resolve it, see your system manager.

4. To test the menu or menus in the directory, enter the following at the UNIX shell prompt:

   ```
   # cd example/inscir
   # CVUI
   ```

   This creates and launches the menus in this directory.

   If the menu has not been programmed to terminate, terminate it manually by using the UNIX “Kill” command or by pressing <Ctrl> C.

Preparing to Use Your Menus in CADDS

1. If the `libUserCr.so` file is not in your current directory, you must modify the `LD_LIBRARY_PATH` (on Solaris) to point to it.

   On Windows, `libUserCR.dll` resides in the `\usr\apl\cadds\bin\` directory which is in the `PATH` environment variable.

2. List the libraries being used to verify that the system is using your local version of `libUserCr.so` instead of `/usr/apl/cadds/slib/libUserCr.so` or `/usr/apl/cvact/slib/libUserCr.so`. Type the following command at the UNIX shell prompt:

   ```
   # cv_ldd `which CR`
   ```

   This is not required for Windows.

3. To avoid having your menus come up automatically when you enter CADDS, change to the directory where your `.act` files and makefiles are located. Edit the file `appinitU.cxx` and comment out the following 2 lines:

```
/*
        st = ClLaunchDialog("InsCircleCab");
        if (HliStatusIsFailure_(st)) return(st);
*/
```

   You must do this for each menu in the directory.

4. To update the libraries when `appinitU.cxx` has been edited, enter the following at the UNIX shell prompt:

```
# make link
```

   On Windows, enter the following command:

```
\usr\apl\cvact\tools\make -f Makelib link
```

   If you have created any new menus, enter the command `share_cvact` instead of `make link`.

Please note: All menus must be built in a single directory.

5. Enter the command `cadds5` at the UNIX shell prompt.

   On Windows, start CADDS from the Start menu.

Please note: Always set the `UI_LOOK_AND_FEEL` environment variable in the `.caddsrc` file on UNIX and the `caddsrc` file on Windows to `CADDS_LAF` so that the menus have the look and feel of CADDS menus.

Using Your Menus in CADDS

1. Choose the Custom option from the pulldown at the task set controller on the left side.
2. Select the Utility option from the top bar and choose Customizer.
3. Drag an empty icon to the edit option in the customizer by pressing the `Ctrl` key simultaneously with the left mouse button.
4. To launch your custom CVact menu, named `InsCircleCab`, first see the file `appinitU.cxx` for the correct name of the menu as seen in line

   st = ClLaunchDialog("InsCircleCab");

   and then move the cursor in front of <`END`> in the Property Editor window, and enter:

```
# launch InsCircleCab
```

5. Click the Apply button from the Property Editor window.

6. Before filing, test the menu by selecting the newly created icon. It should launch your new menu.
7. File the menu using the File Cabinet icon from the Customizer tool.

Platform Specific Information

1. The `share_cvact` script for generating `libUserCr` library is supported on the following platforms:
   - Solaris
   - HP/HP-UX
   - IBM/AIX
   - Windows
2. The `share_cvact` script generates `libUserCr` as a shared library on the following platforms:
   - Solaris
   - HP/HP-UX
   - Windows

Please note: Do not use CADDS 5i or any associated CADDS products on an operating system earlier than that used to build CADDS. All CADDS programming interfaces, including building of customized CVact menus to the `libUserCr` library, is supported only on the CADDS 5i build platforms but can be run on the supported runtime platforms.

3. The `share_cvact` script generates `libUserCr` as a static library on the IBM/AIX platform. In order to use the `libUserCr` static library generated by the `share_cvact` script on the IBM/AIX platform, you are required to build a CR process using the `Makefile` supplied with the CVact distribution.

   Follow the steps outlined to build a CR process on the IBM/AIX platform:

   a. Copy `/usr/apl/cvact/cadds/obj/cr/Makefile` to the `work` directory:

   ```
   cp /usr/apl/cvact/cadds/obj/cr/Makefile /users/guest/
   cvactwork
   ```

   b. Edit the `Makefile` and add the name of the `work` directory containing `libUserCr.a` to the beginning of the line for `LDFLAGS`:

   ```
   LDFLAGS=-L/users/guest/cvactwork...<Existing contents of
   LDFLAGS>
   ```

   c. Make sure `libUserCr.a` exists in the directory specified in step b.

   d. Edit the `Makefile` and change `TARGET` to point to the CR executable in the `work` directory:

```
TARGET =/users/guest/cvactwork/CR
```

e. Edit the `Makefile` and change `OBJSDIR` to point to the `work` directory.

```
OBJSDIR =/users/guest/cvactwork
```

f. Enter the following command at the UNIX shell prompt:

```
# make
```

The CR executable is built in the `work` directory.

4. Create an ASCII file `cadds_init.cfg` with the following contents in the `work` directory which has the CR executable:

```
#
# Configure CR (for running with UIHLI process)
AddTarget CR
ConnectTarget CR null null CR
```

5. Start `cadds5` from the `Work` directory in order to use the CR executable built in the `work` directory with CADDS 5i on IBM/AIX platform.

Please note: Building customized CVact menus to `libUserCr` library is supported only on the CADDS 5i build platforms. Do not attempt to build customized CVact menus to `libUserCr` library on a platform that is not a CADDS 5i build platform. Platforms supported by any CVact version are the same as those supported by the corresponding CADDS version.

6. You cannot use the `CVUI` executable for standalone testing of your menus on the IBM/AIX and Windows platforms.
7. The CVact examples are distributed on IBM/AIX platform as a static library `libUserCr`.a. To build an executable version of the examples, you have to build a CR process as outlined in Step 3 using the appropriate `libUserCr`.a.
8. On an HP platform, the environment variable `SHLIB_PATH` is used for specifying the shared library search path.

Using CVact Menus with CADDS on Windows

With CADDS on Windows, you can compile and use menus designed using CVact on UNIX. The CVact runtime environment allows you to compile and use on Windows the custom CVact menus, message repositories, and the X Pixmap (XPM 2C format) icons.

Design the XPM icons using any standard XPM Icon Editor on UNIX. The CVact Icon Editor that can be accessed on UNIX from `/usr/apl/cadds/bin/iconeditor` is not available on Windows.

You can dynamically load the custom CVact menus for use with CADDS on Windows. These menus are compiled and linked into the shared library `/usr/apl/cadds/bin/libUserCr.dll` that is dynamically loaded in conjunction with a composer replacement (CR) process.

Generate `libUserCr.dll` using the Perl script, `share_cvact.pl`, found in `/usr/apl/cadds/src/data/cvact/tools/NTscripts/`. The CADDS Windows setup program automatically sets the environment path to include the path for Perl and `/usr/apl/cadds/bin`.

Make sure that you have the following software installed on Windows to build the `libUserCr.dll`:

- MKS Toolkit for Enterprise Developers Version 8.6
- MS Visual C++ 6.0

The Perl script processes the custom menus (`.act` files) to include a header file for each menu. The script automatically includes the header file `/cadds/insert/cr/ntdynload/crdllfdef.h` instead of `uihli.h` and `cl.h` in the `cxx` file generated on Windows. These header files declare the function definitions for the UI, HLI, and CL libraries that are used in CR functions and are generated automatically.

Comment out the following lines as shown if you have explicitly entered them in the header section of the CVact menu:

```
#ifndef NUTCRACKER
#include <ui/uihli.h>
#include <hli/hli-core.h>
<cl/cl.h>
#endif
```

Dynamically Loading CVact Menus

After you have installed the NutCracker libraries, you can proceed in creating the CVact menus for dynamic loading. Create a number of custom CVact menus, generate `libUserCr.dll` required to run the custom CVact menus in CADDS in conjunction with a CR process, and start using the menus on Windows using the following steps:

1. Create the CVact menus (`.act` files) on UNIX using the Advanced Customization Tool.

Please note: The CVact interactive tool that is available on UNIX platforms is not available on Windows. The `CVUI` executable used for standalone testing of the custom menus is also not available on Windows.

2. Copy the custom menus (`.act` files) created on UNIX to the `/usr/apl/cadds/lib/UserCR/` directory on Windows.

   The `.act` files must be located in a single directory. On Windows, the directory with the `.act` files must be on the same drive as CADDS.

Please note: Make sure to exit CADDS before you generate the dynamically loaded library `libUserCr.dll`.

3. On Windows, run the Perl script `/usr/apl/cvact/tools/NTscripts/share_cvact.pl` as follows:

```
perl/usr/apl/cvact/tools/NTscripts/share_cvact.pl
```

   The `Makelib` is generated in the directory where the `.act` files are located. The dynamically loaded library `libUserCr.dll` is created in `/usr/apl/cadds/bin` directory.

   The Perl script processes the `.act` files. The `.act` files are compiled and linked into `libUserCr.dll`.

   The CR process automatically picks up `libUserCr.dll` from the directory `/usr/apl/cadds/bin`. You can now start using the custom CVact menus on Windows.

4. Start CADDS on Windows.

   The corresponding menu that you customized on UNIX and ported to Windows NT is dynamically loaded and ready for use in the CADDS session.

Using the Sample Menu

Under the `Example` directory `/usr/apl/cvact/examples` there is a sample created using CVact. Activating the sample menu allows you to see some of the design components described in this book.

The Sample Menu

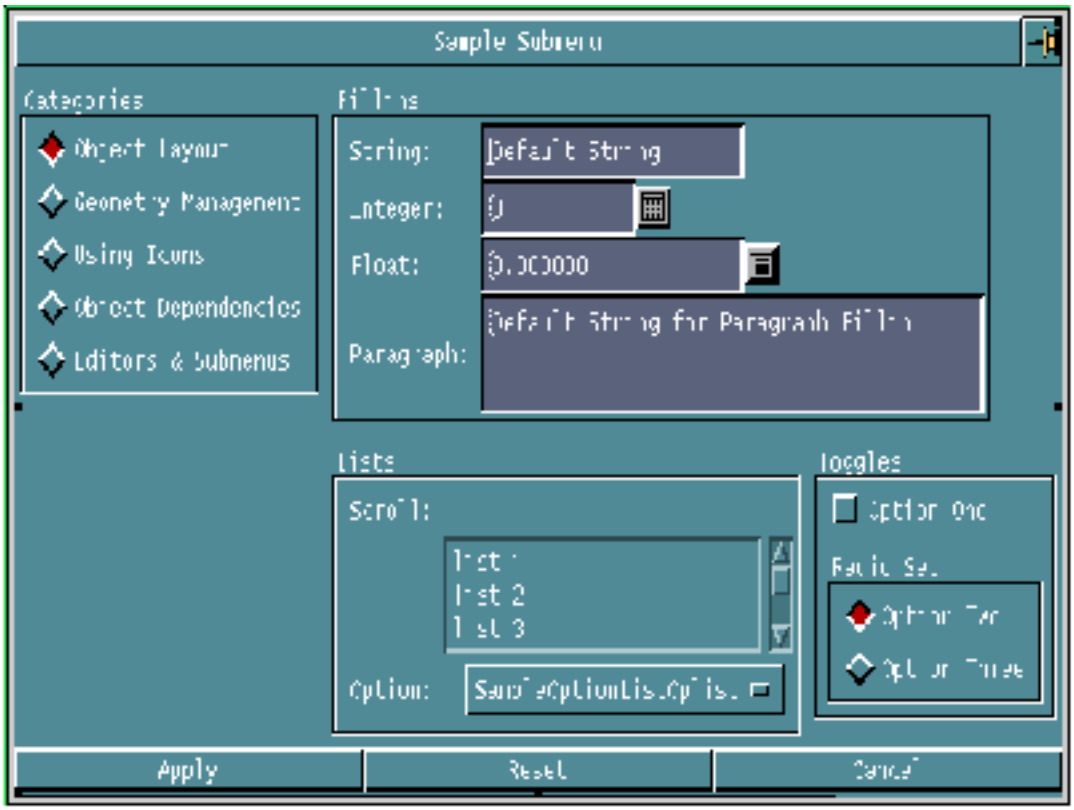
This sample menu presents each feature of CVact according to the company's UI Style Guide and the Motif Style Guide.

In order to activate this menu, enter the following:

```
# source /usr/apl/cvact/Install/cvactrc
# cd /usr/apl/cvact/examples/sample
# CVUI
```

When the menu appears, select widgets and icons to see their effect, modify text and other fill in objects. Click the Apply button to send the result to output.

Figure 4-1 Sample Submenu



Browsing Through the Sample Menu

To browse through the sample menu provided, enter the following:

```
# cd  /usr/apl/cvact/examples/sample
# cvact
```

With CVact active, select File fi Open and open `sample.act`. Click Apply to bring up the main sample menu. You can move throughout the menu to experiment with:

- Callbacks
- Dependencies
- Scroll lists

Geometry Management

You will also see:

- How a radio button is realized
- How the Apply button extracts information from all other objects and passes it to CADDS
- How Apply PullDowns work

Using Editors and Submenus

Open `editorList.act` and `submenu.act`. Select test mode and experiment with the editors and submenus. You will see:

- How editor call back works
- How to launch a submenu

Specifying Initial and Final Codes for .act Files

All `.act` files must have the standard Initial Code (`icode`) and the Final Code (`fcode`) to enable the creation of a single instance of the CVact menu at runtime. Specify the `icode` and `fcode` fragments in the respective sections of the CVact Declarations editor.

The standard Initial Code (`icode`) for a CVact menu is as follows:

```
#ifndef INTERPRETER
static UiObject CreatedCab = NULL;
if (CreatedCab != NULL)
   return(CreatedCab);
#endif
```

The standard Final Code (`fcode`) for a CVact menu is as follows:

```
#ifndef INTERPRETER
CreatedCab = rtrn;
#endif
return(rtrn);
```

If you do not specify the standard Initial and Final Codes for the `.act` files, a new instance of the CVact menu is created with every launch of the menu and the CVact menus may not work correctly.

Customizing Your Menus

If an attribute (see the appendix containing CVact attributes) can be customized by the end user, the letters `cust` in superscript appear after the attribute name, for example, `AlternateIcon`^{cust}. This includes pre-defined values text, label text icons, titles and others.

A Design Rule Customization tool is available to allow you to change customizable attributes while in CADDS or while your menu is running. In order to access the DRC, press the shift key and the right mouse button while the cursor is in a CVact base menu. This will result in launching of the DRC with customizable attributes of the selected object

Figure 4-2 TitleBar

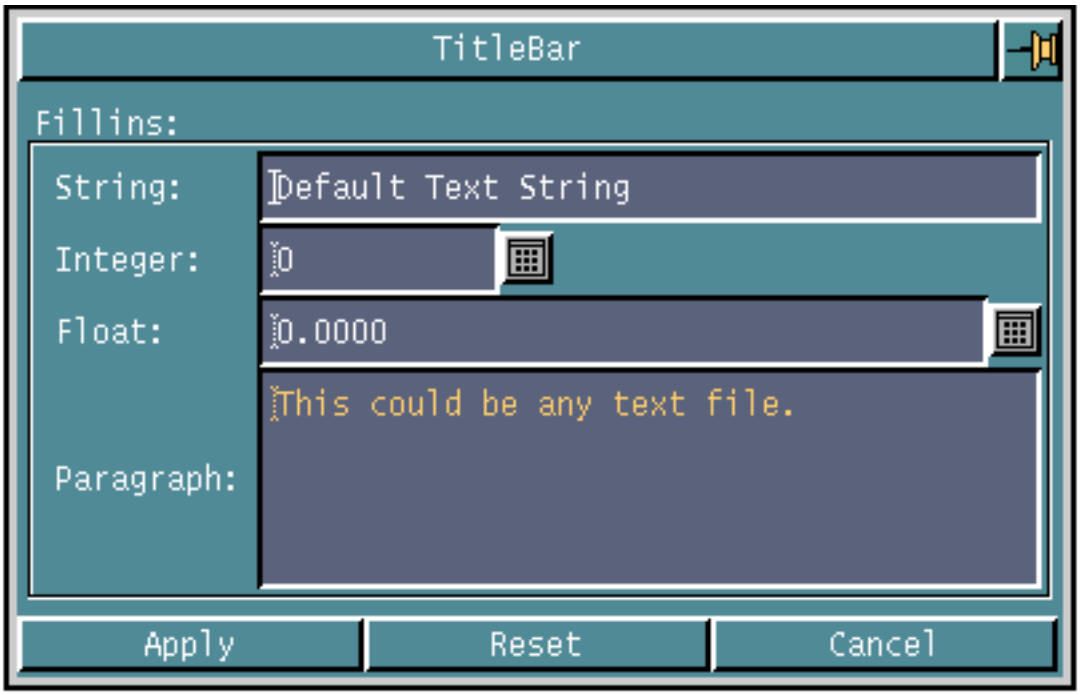
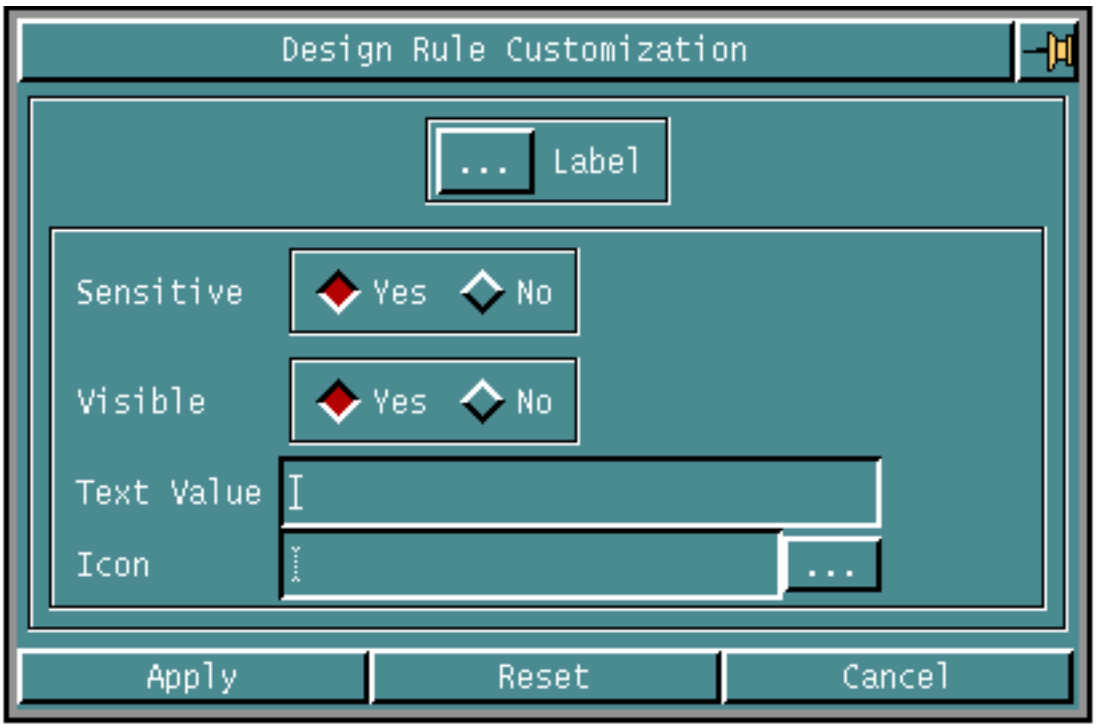


Figure 4-3 Design Rule Customization Tool



In order to edit objects, click the Press To Select Field button. The cursor changes to a circular symbol. This allows you to select any other visible object in order to modify its available attributes. The result will be appended to the file `~/.CR` under your home directory upon selection of the Apply button.

The resulting X resource file can be modified using common editors and is used upon execution of the menus. This method enables any user to customize his or her menus as required.

In order to select existing icons, click the List Button beside the Icon field on the Design Rule Customization dialog box. This allows you to select icons based on the absolute path name or the repository name.

There are two ways by which you can specify the path for the icons:

- @dpi, absolute path name

  To specify the path using the absolute path name, click Absolute Path.
- Repository name, icon name

  To specify the path using the repository name, click Repository Name.

The Icon Browser dialog box is as shown.

Figure 4-4 Icon Browser

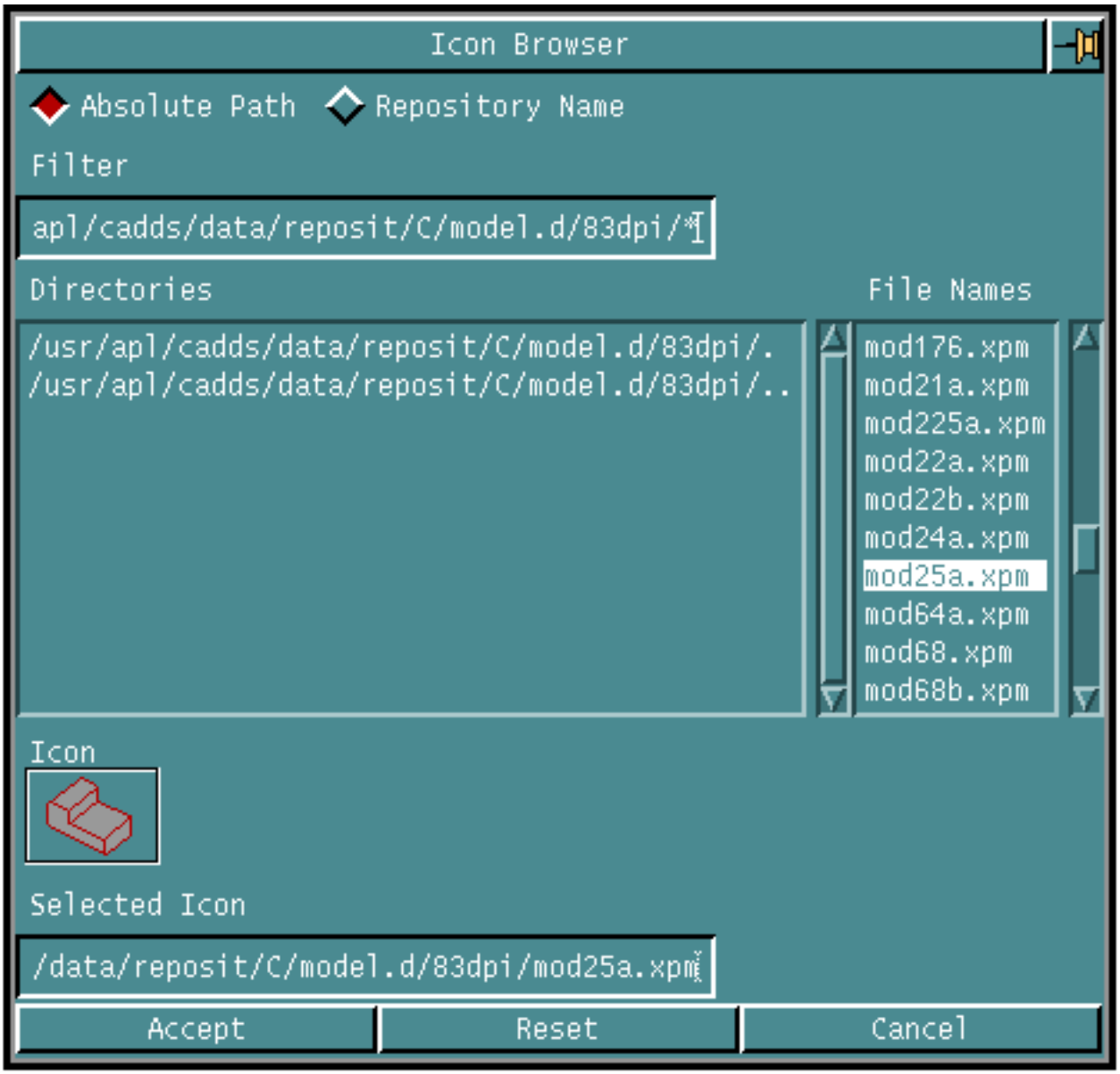
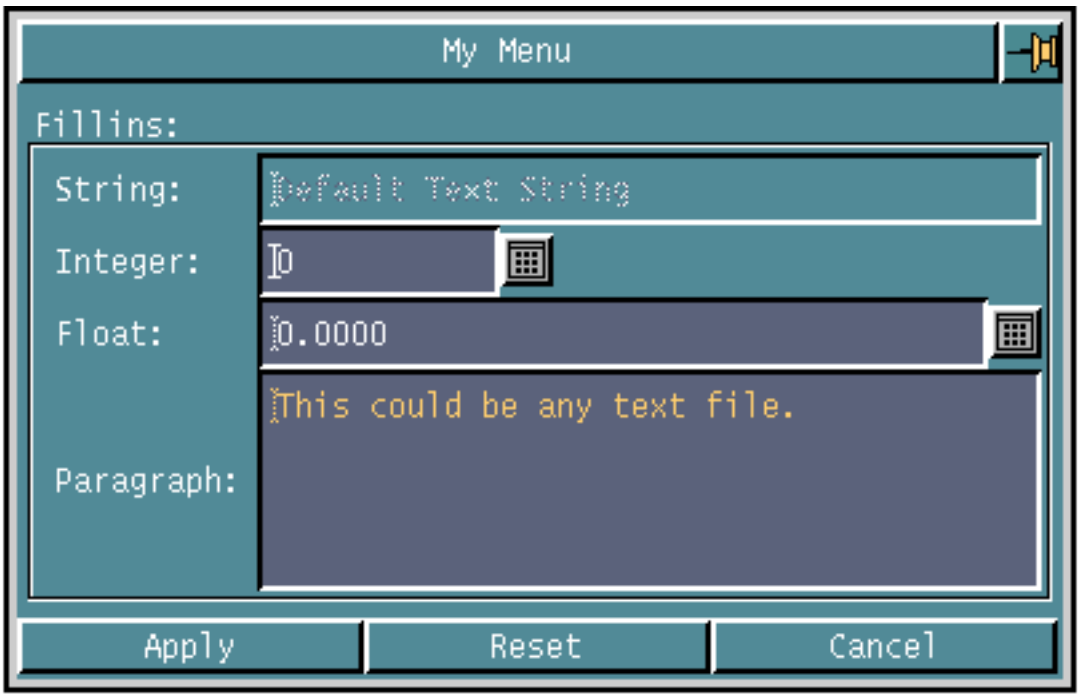


Figure 4-5 A Customized Menu



Sample of the .CR file

```
CR.submenu.title:My Menu
CR.submenu.applyLabel:GO
```

How to Override Customization

System administrators can block access to some or all of the customizable attributes by enforcing their version of the resource file.

Environment `CVUIOVERRIDEPATH` is used as the file locator of the resource file.

Example of Overriding Design Rule Customization

Bring up any CVact menu using CVUI.

When the menu comes up, perform the following steps:

1. Select a label by holding down the shift key while simultaneously selecting with the right mouse button.

   This will bring up the Design Rule Customization Menu for Object Customization.

2. Click `Press To Select Field` button at the top of the menu.

   Notice how the cursor changes to a circle. This means that now you can select one of the objects in your dialog box for editing.

3. Select one of your primitive objects and experiment by modifying its properties.

4. Click Apply when you are satisfied that you want to save your customizations. Note that the customization file is found in your home directory as `.CR`.

5. If you want to use this modified resource file to override the resource file for everyone in your group or department, rename it to `CR`, and place it in the location pointed to by the environment variable `CVUIOVERRIDEPATH`:

```
# cp   /users/you/.CR   /users/you/override/CR  setenv
CVUIOVERRIDEPATH   /users/you/override/CR
```

   It is not possible to modify the field already in the override file.

Behavior of CVact Menus when Iconized or Deiconized

CVact menus of CADDS, when minimized or reduced to icons and then maximized, show the following behavior with respect to field values:

- Fields with remembered values, that is, fields that have the `UI_REMEMBERED` prefill attribute, are restored to the same values that existed just before the menus were minimized or reduced to icons.
- Fields that are prefilled from default, that is, fields that have the `UI_DEFAULT` prefilled attribute, are restored to the default value.
- Fields that have inferred values, that is, fields that have the `UI_INFERRED` prefill attribute, are restored with the recomputed value that gets set when the `InitializeCb` callback is invoked.

The `Prefilled` attribute specifies whether a field has its values obtained by default, remembered from invocation to invocation, or inferred each time the field is displayed.

Please note: If the CVact menu has been tacked, it continues to remain tacked even when you press the `Reset` button on the menu.

Indicating the State of CADDS Desktop

The `Cl_C5_iconized` active variable indicates whether the CADDS desktop is a window or an icon. The settings for the `Cl_C5_iconized` variable are as follows:

- When set to the value of 1, the CADDS desktop is minimized to an icon.
- When set to the value of 0, the CADDS desktop is maximized to a window.

Chapter 5

CVact Attributes

In CVact, an attribute is a variable that determines the properties of a CVact object (or, in X terms, widget). Attributes define the appearance and behavior of objects.

Some examples of attributes are width, height, color, and the activate callback function. You can change these values using CVact's Attribute Editor, and you can set or get most of them via the Ui functions that we provide.

If the attribute can be customized by the end user, we use the letters `cust` in superscript after the attribute name, for example, AlternateIcon^{cust}

- Attribute List

Attribute List

Partial Syntax

Also included are related function calls to get or set the attribute. The majority of these functions have the following syntax:

```
HliStatus UiGetAttributeName
(UiObject obj, AttributeTypeDef AttributeValue)
```

Most Ui object attribute values can be retrieved according to the above syntax, if you know the attribute's name, type, and whether to use a pointer to receive its value or a pointer to the pointer to the value. For example, the `UiGet` and `UiSet` functions for Gutter objects are as follows:

```
HliStatus UiGetGutter(UiObject obj, UiDimension *gutter)
HliStatus UiSetGutter(UiObject obj, HliUnsignedShort gutter)
```

Exceptions to this include objects such as `callbacks`, which take more than two arguments in their get and set functions:

```
HliStatus UiGetCallBackNameCb (UiObject obj,
UiCallbackProc *cb, void **userdata)
HliStatus UiSetCallBackNameCb (UiObject obj,
UiCallbackProc cb, void userdata)
```

Information in this chapter is abbreviated; so a marker is provided to specify Get or Set, followed by the argument signature. For example, for:

- Callbacks

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

- Other attributes, for instance, `ColumnAlignments`

```
Set:(UiObject obj, UiAlignment *colAligns,HliSignedShort num)
```

- Text from message repositories or Hli strings

```
Set...FromMessage (UiObject obj, HliStatus message, ...)
```

AbortCb

ToolEntry:
This callback is executed when an active tool operation is aborted by you. The exact mechanism is platform dependent, but is usually triggered by a key press, such as the escape key.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

Accelerator cust

CommandEntry, ToolEntry, DirectActionEntry, ToggleEntry, SpecifierEntry:

An entry may be assigned an attribute value specifying the keyboard accelerator. When this key is pressed, the effect is as if a related button on the menus or palette was picked. Since the accelerator is a property of the entry, the entry has the same accelerator in all positions where it is referenced.

```
Get:(UiObject obj, UiKey * key)
Set:(UiObject obj, UiKey key)
```

ActivateCb

Entry:
`ActivateCb` is called when the entry is activated, such as being clicked on. For tools, this means picked-up. For commands and direct-actions, this means invoked. For toggles, this means checked/unchecked.

System Data is typically null, except for toggles for which it is the new state of the toggle, a `UiChosenStruct`.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

ActiveWindow

UiObject:
The dataset window which is the current active window.

Alignment

Label, Pushbuttn:
Specifies whether to align text at the start, end, or center of the space allocated by the parent container. (For example, in left-to-right locales, start-alignment means left alignment.) For push buttons, `ui_center_align` is fixed.

```
Get:(UiObject obj, UiAlignment *align)
Set:(UiObject obj, UiAlignment align)
```

AlternateIcon cust

Toggle, Pushbutton :
Normally, an icon is shown to indicate the type of action that a toggle or button performs. An alternate icon shows that the button has been toggled. If both the alternate and normal icon were the same, the user would not know which state the toggle was in.

On Motif platforms, the AlternateIcon is shown when the Icon Toggle is checked. If null, the normal icon is used.

For push buttons, this icon is displayed according to platform conventions, for example, for Motif when the button is armed, when the pointer enters the button, or not at all.

```
Get:(UiObject obj, UiIcon *icon)
Set:(UiObject obj, UiIcon icon)
```

ApplyButton

CommandBox:
If TRUE (default), you can invoke an `ApplyCb` and its associated behavior without dismissing a dialog box. Usually used so that you can keep non-modal boxes posted while doing other things. You can set it to FALSE if the user is expected to dismiss the dialog box, perhaps by pressing an Apply and Ok button, or using the thumb tack.

```
Get:
Set:(UiObject obj, HliBoolean state)
```

ApplyCb

CommandBox :
Called when the OK button or the Apply button is clicked, to dismiss a dialog box.

System Data is a transient pointer to a Boolean value that tells whether the dialog box is confirmed when this callback returns. The value is initially TRUE, but the callback is free to change this value to FALSE to indicate that there was a problem and the command could not be performed, for example.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

ApplyLabel cust
ApplyLabelFromMessage
ApplyLabelFromVMessage

AlignCmdBox, CommandBox:
The text is used to label the Apply button, which confirms without dismissing the dialog box. If your application does not set this attribute, or sets it to NULL, the default label will be used.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliShortString buf)
Set:(UiObject obj, HliConstShortString buf)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

ArgumentBox

CommandEntry:
The `CommandEntry` has an attribute, `ArgumentBox`, which is the command argument box to be launched. If this entry is non-null, the launching is automatic. The `ArgumentBox` attribute will be used to look up the corresponding command argument box by name and will be made visible if it exists. If it does not exist, the command entry's `ActivateCb` will be called to create it and make it visible.

```
Get:(UiObject obj, const char ** argumentBox)
Set:(UiObject obj, const char *argumentBox)
```

AssistLine cust
AssistLineFromMessage
AssistLineFromVMessage

UiObject:
Text shown in the assistance area when mouse or cursor lies on top of this object. You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliShortString assist)
Set:(UiObject obj, HliConstShortString assist)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

AssistOn cust

ApplicationDesktop:
A Boolean attribute that indicates whether the assist area is to be visible.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

Background

BuildingBlock:
Background color category for an object. If set to null, the default for the object class is used.

```
Get:(UiObject obj, UiColor * background)
Set:(UiObject obj, UiColor background)
```

ButtonParent

CommandBox :
The object into which application-created push buttons should be parented.

```
Get:(UiObject obj, UiObject *bp)
```

CancelCb

CommandBox :
Called when the Cancel button is pressed, for example, to close a file that may have been opened when the CAB was invoked. After the callback returns, the dialog box is dismissed.

In the case of a `WarningBox`, the Field will be reset. In the case of a `QuestionBox`, this answers “No” to the question. In the case of a `Working Box`, this callback should be used to interrupt the application.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

CancelLabel
CancelLabelFromMessage
CancelLabelFromVMessage

CommandBox :
Text used on the Cancel button for a Command Box. You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliShortString buf)
Set:(UiObject obj, HliConstShortString buf)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

Cell

ScrollMatrix:
Text that can be set or got for an individual cell.

```
Get:(UiObject obj, row, col, *val)
Set:(UiObject obj, HliSignedShort row, HliSignedShort col,
HliConstString val)
```

CellValues

ScrollMatrix :
Two dimension array of values in the cells of the matrix. Each row is an array of Hli strings, one for each column in that row. If null, will be automatically filled with `hli_null_strings`. When set, the dimensions of the array must match the existing `RowsNum` and `ColumnsNum`.

```
Set:(UiObject obj, HliString *const cellValues[ ],
HliSignedShort rows, HliSignedShort columns)
```

Checked cust

Toggle :
If TRUE, the toggle is checked; otherwise it is unchecked. When changed, the new value is in place before the `ActivateCb` is made.

Since this is an attribute you can customize, your application should get this attribute immediately after creating the toggle, to see what the initial state of the toggle really is.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

Children

Container:
The instance children, whether managed or not, excepting only those which have been destroyed.

```
Get:(UiObject obj, UiObject children[])
```

ChildrenNum

Container:
The number of items or components in Children.

```
Get:(UiObject obj, HliSignedShort *num)
```

Chosen

ScrollList:
A set-only attribute that lets your application specify whether an element in a scroll list has been chosen. Setting to TRUE (to indicate chosen) activates scrolling if the item is not visible, but does not activate a call to the ChosenCb.

```
Set:(UiObject obj, HliSignedShort idx, HliBoolean chosen)
```

ChosenCb

RadioSet, Toggle, ScrollList, OptionList :
Issued when the state is changed, for example, for a single click on an item. This callback is not called when a value is explicitly set using `UiSet`; but for toggles, scroll lists, and option lists, is it called whenever an item is set internally from a default or remembered value.

For radio sets, this provides the default value for the `ChosenCb` attribute of each child. (The child normally calls back when it becomes chosen or not chosen.)

For toggles, System Data is a transient pointer to the new value of Checked.

For scroll lists, System Data is a transient pointer to TextValue. It is a transient UiChosenStruct that contains item string, item position, new chosen value : TRUE or FALSE, and modst : modifiers state. `Modst` is a bitmap with each bit

representing an X toolkit Intrinisics (Xt) modifier. Only the Keyboard `Shift` key modifier represented by the first right most bit is used. Use the mask `UI_SHIFT_MOD_MASK` to test the state of the keyboard `Shift` key modifier. For option lists, System Data is a transient pointer to TextValue.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

ChosenItems

ScrollList :
Array of zero-based position numbers showing which items are currently chosen.

```
Get:(UiObject obj, HliSignedShort *itms)
```

ChosenItemsNum

ScrollList :
Number of values in `ChosenItems` array.

```
Get:(UiObject obj, HliSignedShort *numItms)
```

ChosenObject

RadioSet :
This is the object that is the current choice. If set to null then none is chosen (if the platform permits; otherwise, the first sensitive visible object is chosen).

```
Get:(UiObject obj, UiObject *c)
Set:(UiObject obj, UiObject c)
```

ClosedCb

TopLevel, DialogBox :
Performed when you close (for example, terminates or iconifies) this object.

For `DialogBox`, called when you cancel or confirm the box, or use window manager functions to dismiss it. Field contents must be confirmed before the `ClosedCb` is called. System Data is a transient pointer to the boolean TRUE if the dismissal came from the window manager.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

Column Alignments

ScrollMatrix :
An array of text alignments, one for each column, that specify whether that column's text should be left-, center-, or right-aligned.

```
Get:(UiObject obj, UiAlignment *columnAlignments)
Set:(UiObject obj, UiAlignment *colAligns, HliSignedShort num)
```

ColumnLabel

ScrollMatrix:
The text for the labels above a column. See `ColumnLabels`.

```
Get:(UiObject obj, HliShortString colLabel, HliSignedShort idx)
Set:(UiObject obj, HliShortString colLabel, HliSignedShort idx)
```

ColumnLabels

ScrollMatrix :
An array of labels to be drawn above each column. Embedded newline characters are permitted and allow the label to be drawn on multiple lines.

```
Get:(UiObject obj, HliString *columnLabels)
Set:(UiObject obj, HliString *columnLabels, HliSignedShort num)
```

ColumnWidths

ScrollMatrix :
Array of widths, one for each column, specifying the visible width (in characters) of the cells in the column. May not be null.

```
Get:(UiObject obj, HliSignedShort *colWids)
Set:(UiObject obj, HliSignedShort *colWids, HliSignedShort num)
```

ColumnsNum

ScrollMatrix :
Number of columns in the cell. Must be at least one.

```
Get:(UiObject obj, HliSignedShort *columnsNum)
```

ColumnsWide

Field:
This attribute specifies the number of columns that this field will occupy in an aligning form (zero or negative numbers are changed to 1). This value may be reduced if it extends the field beyond the number of columns provided by the aligning form.

This attribute may only be referenced if this field is an immediate child of an aligning form.

```
Get:(UiObject obj, HliSignedShort* wide)
Set:(UiObject obj, HliSignedShort wide)
```

CommandOn cust

ApplicationDesktop:
A Boolean attribute that indicates whether the command area is to be visible.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

CompleteCb

Fillin, ScrollList, OptionList, Paragraph :
Called when focus leaves the object, for example, for a field validation check. Focus can leave an object when the user completes a field (for example, by tabbing, trying to confirm the parent dialog box contents, or selecting an option in a option list) or when a value is loaded from default.

For fillin fields, this callback will validate the current `TextString` and set or clear the Defective attribute. System Data is a transient pointer to the TextString.

For option lists, System Data is a transient pointer to `TextValue`.

For scroll lists, System Data is a transient pointer to a `UiCompleteStruct`, which contains (ChosenItems, ChosenItemsNum).

For paragraphs, this callback should validate the current `TextValue` and return FALSE if it is unacceptable. The Defective attribute is automatically set or cleared in the appropriate manner after the callback returns. System Data is a transient pointer to the `TextString`.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

For the Fillin, IntFillin, FloatFillin, and Paragraph fields, the CompleteCb callback is called when the field loses keyboard focus in the following instances:

- Click in any other Fillin, IntFillin, FloatFillin, or Paragraph field.
- Click outside the CVact property sheet.
- Press the `Tab` key or press the `Shift` and `Tab` keys together (`Shift` key simultaneously pressed with the `Tab` key) on the keyboard to move the keyboard focus to the next or previous Fillin, IntFillin, FloatFillin or Paragraph field.
- Click `Apply` in the Command Align Box (CAB) after modifying a field.
- Press `Carriage Return` after completing a field.

Please note: Keyboard focus is different from mouse focus. A field has keyboard focus if it can accept or receive any character entered from the keyboard. The field is said to have lost keyboard focus when it can no longer accept or receive any character entered from the keyboard.

CompleteCellCb

ScrollMatrix :
This callback is called after a cell has been edited. Presumably the callback will validate the value and set the “doit” value in the structure to FALSE if the new value is unacceptable. In this case, the new value is discarded and the cell retains its original value. System Data is transient pointer to an un-editable text string in a `UiCompleteCellCallbackStruct`.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

ConstantHeight

AligningForm :
TRUE means that all rows must be the same height (but rows without contents can still be zero height). FALSE allows each row to have a different height, computed on the basis of its contents.

```
Get:(UiObject obj, HliBoolean* state)
Set:(UiObject obj, HliBoolean state)
```

ConstantWidth

AligningForm :
TRUE means that all columns must be the same width (except those without contents, which can have zero width.) FALSE lets each column have a different width, computed on the basis of its contents.

```
Get:(UiObject obj, HliBoolean* state)
Set:(UiObject obj, HliBoolean state)
```

Contents cust

EntryList:
An array of `Entries` and `EntryLists`. This attribute is used in resource files to specify the contents of an EntryList. For example, select `Contents= Select`, Deselect specifies that select is the EntryList that contains the Entries Select and Deselect**.**

```
Get:(UiObject obj, UiObject contents[])
```

ContentsNum

EntryList:
Number of entries in the Contents.

```
Get:(UiObject obj, HliUnsignedShort * num)
```

CurrentTool cust

ApplicationDesktop:
The Toolentry which is the current tool in use. When the application desktop is first created, a value is placed here which becomes the first tool activated.

Cursor

ToolEntry:
The cursor that is to be used in the presentation area from the time that a tool is activated until it is deactivated. (The presentation area contains CADDS drawings, for example.)

```
Get:(UiObject obj, UiCursor *cursor)
Set:(UiObject obj, UiCursor cursor)
```

CursorPosition

Fillin, ScrollList, Paragraph :
For fillin and paragraph fields, the cursor lies in front of the character designated by this value. If the application sets this attribute, the cursor moves. Characters are numbered from zero.

For paragraphs, scrolling is performed so that the cursor is in the visible portion of the field. For scroll lists, tells what item the cursor is on (again, zero-based). Setting this attribute moves the cursor and may result in scrolling.

```
Get:(UiObject obj, HliSignedShort *curPos)
Set:(UiObject obj, HliSignedShort curPos)
```

DeactivateCb

ToolEntry:
This callback is performed when the user drops a tool, even if he reselects the same one. The `DeactivateCb` is called before the next tool becomes active.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

DecimalPlaces cust

FloatFillin:
The number of places to the right of the decimal that are to be shown when values are converted from `FloatValue` (see p. x in this chapter). The user may enter more or fewer decimal places. This only restricts the number of decimal places when the `FloatValue` is converted from `TextString`, not the number that may be contained in `TextString`.

```
Get:(UiObject obj, HliSignedShort *value)
Set:(UiObject obj, HliSignedShort value)
```

DefaultActionCb

ScrollList :
Called when an item is invoked, for example, double-clicked, for default action. Platforms that do not support this operation ignore this callback. System Data is a transient pointer to a structure consisting of the item string and item position.

For scroll lists, System Data is a transient UiChosenStruct that contains item string, item position, new chosen value : TRUE or FALSE, and modst : modifiers

state. `Modst` is a bitmap with each bit representing an X toolkit Intrinisics (Xt) modifier. Only the Keyboard `Shift` key modifier represented by the first right most bit is used now. Use the mask `UI_SHIFT_MOD_MASK` in order to test the state of the Keyboard `Shift` key modifier.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

DefaultBoolean cust

Toggle :
The default value. If `Prefilled`=UI_DEFAULT_PREFILLED, this value is copied into Checked at prefill time.

```
Get:(UiObject obj, HliBoolean *value)
Set:(UiObject obj, HliBoolean value)
```

DefaultButton

DialogBox :
One of the child push buttons, this button is the default push button (on platforms that support this concept). A value of null defaults to the first button.

```
Get:(UiObject obj, UiObject *db)
Set:(UiObject obj, UiObject db)
```

DefaultFloatValue cust

FloatFillin :
Whenever this value is set, it automatically sets the `DefaultString` value, converted according to locale-specific rules.

```
Get:(UiObject obj, HliDouble *value)
Set:(UiObject obj, HliDouble value)
```

DefaultIntegerValue cust

IntegerFillin:
Whenever this value is set, it automatically sets the `DefaultString` value, converted according to locale-specific rules.

```
Get:(UiObject obj, HliSignedLong *value)
Set:(UiObject obj, HliSignedLong value)
```

DefaultItems

ScrollList :
The default list of chosen data. An array showing items that are to be chosen by default. If `Prefilled=UI_DEFAULT_PREFILLED`, this array will be copied into `ChosenItems` at prefill time.

```
Get:(UiObject obj, HliSignedShort *Items)
Set:(UiObject obj, HliSignedShort *Items, HliSignedShort num)
```

DefaultItemsNum

ScrollList :
Number of items in `DefaultItems` array.

```
Get:(UiObject obj, HliSignedShort *dfltItemsNum)
```

DefaultObject

RadioSet:
For `RadioSets`, the name of the default toggle choice for the radio set. The toggle will be made the `ChosenObject` at prefill time if `Prefilled= ui_default_prefilled`. If `DefaultName` is null, a toggle with `DefaultBoolean=true` wishes to be the default choice. The last-created toggle that has `DefaultBoolean=true` becomes the `ChosenObject`. If no toggle is default, then the first toggle becomes the `ChosenObject`.

```
Get:(UiObject obj, const char* *d)
Set:(UiObject obj, const char* d)
```

DefaultString cust
DefaultStringFromMessage
DefaultStringFromVMessage

Fillin, Paragraph :
The default text; it will be copied into `TextString` at prefill time, if `Prefilled = ui_default_prefilled`.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliString value)
Set:(UiObject obj, HliConstString value)
```

```
Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

DefaultStringNum

Fillin :
Number of characters in the Default String.

```
Get:(UiObject obj, HliSignedShort *num)
```

DefaultValue cust

ScrollList, OptionList :
For scroll lists, a string showing a desired default value. When UiAdditem(s) is used, the added items are compared against `DefaultValue`; if there is a match, that item's position is recorded in the `DefaultItems` array.

For option lists, if `Prefilled= ui_default_prefilled`, this value will be copied into `TextValue` at prefill time.

```
Get:(UiObject obj, HliShortString value)
Set:(UiObject obj, HliConstShortString value)
```

Defective

Fillin :
TRUE means that the contents of the field have been deemed unacceptable, presumably having failed validation performed by the `CompleteCb`.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

Definer

Entry:
A class variable that holds the address of the application entry definer.

Dependency

UiObject:
Used to specify which attribute (sensitive, managed, or visible), is affected by the DependencyExpression. See `UiDependency`.

```
Get:(UiObject obj, UiDependency *type, const char **expression)
Set:(UiObject obj, UiDependency type, const char *expression)
```

DependencyExpression

Field:
Null-terminated character string containing the dependency expression.

```
Get:(UiObject obj, const char* *expression)
Set:(UiObject obj, const char *exp)
```

DependencyType

Field:
Tells whether the dependency expression controls the `Sensitive`, `VisibleWhenManaged`, or `Managed` attribute (or none of them).

If the `DependencyType` attribute is non-null, the `DependencyExpression` is continuously evaluated to control the `Sensitive`, `VisibleWhenManaged`, or `Managed` attribute.

```
Get:(UiObject obj, UiDependency* type)
Set:(UiObject obj, UiDependency type)
```

DestroyCb

UiObject:
This callback is invoked when the object is about to be destroyed. As soon as the callback returns, the object will be destroyed. System Data is `hli_null_system_data`. This callback is not performed during “fast shutdown.”

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

DropCb

Field:
A non-null value allows this field to be a drop target.

If a drag operation is: (a) in progress (using a Drag Form), and (b) you position the drag form over this field, and (c) the `DropCb` is non-null, then the field visually distinguishes itself to show that dropping here is permitted. If you drop here, the

DropCb is performed. SysData is a transient pointer to the UiObject which is the drag form that was dropped.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

Editor

Fillin, IntegerFillin, FloatFillin :
If not null, the name of this field's Editor Dialog Box. This Editor Dialog Box provides the icon used for the Editor Icon.

```
Get:(UiObject obj, const char **editor)
Set:(UiObject obj, char* editor)
```

EditorAttributes

Fillin, IntegerFillin, FloatFillin:
A set of attributes affecting the icon and behavior of an Editor Dialog Box such as the calculator one gets with an Integer or Float Fillin. The attributes are: Input, EditorIcon, EditorInvokedCb, EditorRescindCb, and EditorFocusChangedCb.

```
Set:(UiObject obj, UiIcon icon,
UiCallbackProc invokeProc, void *invokeData,
UiCallbackProc rescindProc, void *rescindData,
UiCallbackProc focusProc, void *focusData)
```

EditorFocusChangeCb

DialogBox :
This callback is performed when an associated fillin field gets the focus of attention, even if the Editor Dialog Box is invisible or not in use. System Data is a transient pointer to the UiObject representing the associated field.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:UiObject obj, UiCallbackProc cb, void* userdata)
```

EditorIcon

DialogBox :
This icon is supplied if the dialog box is to be an Editor Dialog Box. Any fillin field that associates this dialog box as its Editor will display this icon as the Editor Button for the user to press.

```
Get:(UiObject obj, UiIcon *ico)
Set:(UiObject obj, UiIcon ico)
```

EditorInvokedCb

DialogBox :
Invoked when you press the Editor Button in an associated fillin field, whether the Editor Dialog Box is visible or already in use. System Data is a transient pointer to the UiObject that represents the associated field.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

EditorRescindCb

DialogBox :
The callback to perform when a dialog box containing an associated fillin field is closed, even if the Editor Dialog Box is invisible or not in use.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

Entry

Menu:
Typically, the `TextValue` is shown if the entry is in a menu. The text of menu labels is rendered using UI_MENU_FONT.

```
Get:(UiObject obj, HliSignedShort pos, UiObject *entry)
```

FloatValue

FloatFillin :
This is the floating-point representation of the `TextString`. When `FloatValue` is set, the `TextString` is set to match, and vice versa, so that they are always equivalent. `Conversion` from `FloatValue` to `TextString` is rounded off to `UiDecimalPlaces`, independent of locale.

Please note: It is possible that the `FloatValue` is not in the range specified by the `MinFloatValue` and `MaxFloatValue`. This would be the case, for example, if the Defective attribute were TRUE, or if you were in the process of inputting values into the field.

```
Get:(UiObject obj, HliDouble *value)
Set:(UiObject obj, HliDouble value)
```

FontCategory

Fillin, Label, ScrollList, OptionList, Paragraph :
Specifies the font category to be used to display text in an object. Setting to null means to accept the default FontCategory for this object class.

For scroll and option lists, this specifies the font category for the items in the list.

```
Get:(UiObject obj, UiFontCategory *font)
Set:(UiObject obj, UiFontCategory font)
```

Foreground

BuildingBlock:
Foreground color category for an object. If set to null, the default for the object class is used.

```
Get:(UiObject obj, UiColor * foreground)
Set:(UiObject obj, UiColor foreground)
```

Framed

Container, Field, NonAligningForm, RadioSet:
Specifies whether to display a frame around the object.

For Container, if true, a visible frame is drawn around the entity, and the margin (extra width and height) is increased to accommodate the size of the frame.

For non-aligning forms, fixed at FALSE.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

Gutter

Aligning Form:
The space between children of an aligning form. Set to 0, there will be no space.

```
Get:(UiObject obj, UiDimension *gutter)
Set:(UiObject obj, HliUnsignedShort gutter)
```

Height

BuildingBlock:
The current height of the building block. These values are only guaranteed to be correct if `VisibleNow` is TRUE. The height of the building block includes any border or margin that the building block itself provides, but excludes a border or margin provided externally (for example, by the parent or by the platform's window manager).

You can use the convenience functions `UiGetGeometry` for the X, Y position, and the width and height.

```
Get(UiObject obj, UiDimension *height)
```

HeightSpecified

BuildingBlock, DialogBox:
This is the application's desire for the box height, which the parent's geometry manager may or may not honor. The units are in UiDimensions.

For dialog boxes, the height excludes window decorations. For non-aligning forms, the size stays the same.

The preferred minimum size implies the default value, which depends on the object's contents, their font, and so on. You can use the convenience functions `UiGetGeometrySpecified` or `UiSet GeometrySpecified` for the X, Y position, and the width and height.

```
Get:(UiObject obj, UiDimension *height)
Set:(UiObject obj, UiDimension height)
```

HelpContext

UiObject, BuildingBlock:
Help Context descriptor passed to the Help Subsystem if the user asks for help on this object.

For Buildingblock, value `ui_null_help_context` means that the help context of the Parent is to be used.

```
Get:(UiObject obj, UiHelpContext *ctx)
Set:(UiObject obj, UiHelpContext help)
```

HLS cust

ColorCategory:
A color attribute specified in Hue/Lightness/Saturation (HLS) values.

```
Get(UiObject obj, UiColor color, HliDouble *hue,
HliDouble *lgt, HliDouble *sat)
Set(UiObject obj, UiColor color, HliDouble hue, HliDouble lgt,
HliDouble sat)
```

Icon cust

(Label) :
This is the icon that will be displayed on an object.

```
Get:(UiObject obj, UiIcon *icon)
Set:(UiObject obj, UiIcon icon)
```

IconifiedIcon cust

TopLevel:
The icon to be displayed if the object is iconified. If `ui_null_icon`, the default icon is used, subject to the current window manager.

```
Get(UiObject obj, UiIcon *icon)
Set(UiObject obj, UiIcon icon)
```

IconifiedTitle cust
IconifiedTitleFromMessage
IconifiedTitleFromVMessage

TopLevel:
The title to be displayed under the icon if the object is iconified.
`hli_null_string` means to use the window title.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliString title)
Set:(UiObject obj, HliConstString title)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

InitializeCb

Field, ScrollList :
Issued when an object is going to be reset or made visible.

For Field, invoked whenever the field is initialized or reset (for example, each time the parent Dialog Box becomes visible). This callback is called after the field is prefilled from default or remembered values (if appropriate) and before the field is actually displayed. System data parameter is null.

For ScrollList, since `Prefilled=inferred`, this callback is called the first time the scroll list is about to be displayed. Presumably, at the first invocation, this callback will place items into the list, perhaps using `UiAddItem`(). The callback may also set Prefilled to `ui_remembered` or `ui_default_prefill`.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

Inline cust

EntryList:
This Boolean attribute for an `EntryList` signifies that the entries in the list are to appear inline in the current menu and not in a separate menu. Thus they are accessed from the current menu rather than by pulling right to another hierarchical level, as you would do for a cascading menu.

```
Get:(UiObject obj, HliBoolean * state)
Set:(UiObject obj, HliBoolean state)
```

InputPermitted

Fillin, Paragraph :
If TRUE, then input is permitted. If FALSE, attempts to change the value do not work. Either way, outbound clipboard operation (Copy) and drag of contents can still be performed, subject to platform restrictions. However, if FALSE, no user edit operation is permitted to change `TextString`.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

The visibility of the cursor corresponds to the InputPermitted state, whether set to TRUE or FALSE. Fillin, IntFillin, FloatFillin, and Paragraph fields do not show the cursor when the InputPermitted attribute is set to FALSE.

IntegerValue

IntegerFillin :
The integer representation of the `TextString`. When either IntegerValue or `TextString` is set, the other is set to match. They are always equivalent.

```
Get:(UiObject obj, HliSignedLong *value)
Set:(UiObject obj, HliSignedLong value)
```

IsEditableCb

ScrollMatrix :
This callback is called immediately before a cell is to be edited, presumably to decide if it is permissible to edit the cell, and, if not, to set the "doit" value in the structure to FALSE so that the user can not add or delete characters. System Data is a transient pointer to `UiIsEditableCallbackStruct`.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

IsInputOkCb

Fillin, IntegerFillin, FloatFillin, Paragraph, ScrollMatrix :
Issued when the contents of an object have changed because you have typed or deleted a character. It can also be used to suppress the character echo when a password is input. System Data is a transient pointer to a structure containing the length of the proposed revised `TextString` followed by the null-terminated string itself.

Usually, this callback validates the current `TextString`, and, if it is unacceptable, changes the length in the System Data to -1, causing the proposed new `TextValue` to be rejected. It may change the string, as long as it doesn't become longer and is still terminated with `hli_null_char`.

It is not called for actions like moving the cursor, that do not change `TextString`, nor when the application explicitly sets the value of `TextString`, nor when the value is set internally by copying from the default or remembered values.

For fillin fields, this is called immediately and every time you change the `TextString` (for example, by typing a character, deleting a character, or pasting).

For scrolled matrixes, this is called after each edit inside a cell. System Data is a transient pointer to `UiCellInputOkCallbackStruct`, which includes a UiIsInputOk structure containing the length of the new string and the string.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

Item

ScrollList, OptionList:
The text value of an item in a scroll or option list.

```
Get:(UiObject obj, HliSignedShort idx, HliShortString str,
HliBoolean *chosen)
```

ItemsNum

ScrollList, OptionList :
The number of items in a scroll list, or the number of strings in items found in option lists.

```
Get:(UiObject obj, HliSignedShort *numItms)
```

Use option lists for lists with 10 items or fewer. Use scroll lists instead of option lists for lists with more than 10 items.

LabelFontCategory

ScrollMatrix :
The font category to be used for displaying row and column labels.

```
Get:(UiObject obj, UiFontCategory *font)
Set:(UiObject obj, UiFontCategory font)
```

Manage

BuildingBlock:
TRUE means that this building block participates in geometry management of its parent and is therefore eligible to be visible. If FALSE, the building block will not participate in geometry management and therefore can't be visible.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

MaxFloatValue cust

FloatFillin :
The maximum value permitted for `FloatValue`, for validation. If the user enters a larger value, the field is marked and displayed as defective.

```
Get:(UiObject obj, UiObject obj,HliDouble *value)
Set:(UiObject obj, HliDouble value)
```

MaxIntegerValue cust

IntegerFillin :
The maximum value permitted for `IntegerValue`, for validation. If you enter a larger value, the field is marked and displayed as defective.

```
Get:(UiObject obj, HliSignedLong *value)
Set:(UiObject obj, HliSignedLong value)
```

MaxLength

Fillin, IntegerFillin, FloatFillin, Paragraph :
For fillin fields, the maximum length, in characters, that you are permitted to enter.

For integer fillin fields, the default value is computed based on the number of digits that may be supplied and the possible presence of thousands-separators, all as determined by `MaxIntegerValue` and `MinIntegerValue`.

For float fillin fields, the default value computation also considers decimal separators and exponent, as well as `MaxFloatValue` and `MinFloatValue`.

For paragraphs, the maximum number of characters of the total paragraph contents. You will not be permitted to enter more than this many characters.

```
Get:(UiObject obj, HliSignedShort *max)
Set:(UiObject obj, HliSignedShort max)
```

MaxVisibleLength

Fillin, Paragraph:
For fillins, the preferred maximum number of visible characters. (Remaining characters can be seen by scrolling.) This preference might not be honored, depending on size available in the parent.

For paragraphs, the preferred maximum number of characters that should be visible per line.

```
Get:(UiObject obj, HliSignedShort *max)
Set:(UiObject obj, HliSignedShort max)
```

MaxVisibleLines

Paragraph :
The preferred maximum number of lines that should be visible (remaining characters can be seen by scrolling). This preference might not be honored, depending on the size available in the parent.

```
Get:(UiObject obj, HliSignedShort *max)
Set:(UiObject obj, HliSignedShort max)
```

MinFloatValue cust

FloatFillin :
The minimum value permitted for `FloatValue`. If the user enters a smaller value, the field is marked and displayed as defective.

```
Get:(UiObject obj, UiObject obj,HliDouble *value)
Set:(UiObject obj, HliDouble value)
```

MinIntegerValue cust

IntegerFillin:
The minimum value permitted for `IntegerValue`. If the user enters a smaller value, the field is marked and displayed as defective.

```
Get:(UiObject obj, HliSignedLong *value)
Set:(UiObject obj, HliSignedLong value)
```

MinVisibleColumns

ScrollMatrix :
The preference for the number of columns to be visible at a time.

```
Get:(UiObject obj, HliSignedShort *minVisCols)
Set:(UiObject obj, HliSignedShort minVisCols)
```

MinVisibleLength

Fillin, Paragraph :
For fillin fields, the preferred minimum number of visible characters. (Remaining characters can be seen by scrolling.) This preference might not be honored, depending on size available in the parent.

For paragraphs, the specified line-length, in number of characters, used only to determine the preferred visible width of the field. Lines can be shorter. Longer lines are allowed and scrolled.

```
Get:(UiObject obj, HliSignedShort *min)
Set:(UiObject obj, HliSignedShort min)
```

MinVisibleLines

Fillin, ScrollList, Paragraph :
Recommendations on how many items should be shown. Used to compute the preferred minimum height of the list or field.

For paragraphs, the specified number of lines to display. If there are more lines than this, they can be scrolled.

```
Get:(UiObject obj, HliSignedShort *minVisLines)
Set:(UiObject obj, HliSignedShort minVisLines)
```

MinVisibleRows

ScrollMatrix :
The preferred number of rows that should be visible at a time.

```
Get:(UiObject obj, HliSignedShort *minVisRows)
Set:(UiObject obj, HliSignedShort minVisRows)
```

Modal

DialogBox :
TRUE means that the dialog box is application modal, i.e., the user cannot work elsewhere in the application without dismissing the dialog box, but may work in another application on some platforms.

```
Get:(UiObject obj, HliBoolean *m)
```

MouseEventCb

ToolEntry:
This callback is invoked when a mouse event occurs which is enabled in the `MouseEventMask` attribute. System data is an aggregate of data associated with the mouse event.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

MouseEventMask

ToolEntry :
A bit mask which defines the class of mouse events which will trigger a mouse event callback. This includes button events, motion events, enter/leave events, etc.

Name

UiObject, ApplicationState, Entry, Color, FontCategory, ApplicationDesktop :
The absolute name of this object instance, as it will be referred to in scripts. This name is sometimes user-visible so it should be meaningful to the user (avoid obscure naming).

For UiObjects, Name must be set at creation and only then.

For `ApplicationState`, Name must not collide with any other `ApplicationState` or top-level objects or Entries.

For Color, Name must not collide with any other `ApplicationState` or top-level objects or Entries. For Entries, the name must be unique across all entries, `EntryLists`, and `TopLevel` objects and should not reflect the position in the menu or palette hierarchy, because those might change. For `FontCategory`, Name must not collide with any other `ApplicationState` or top-level objects or Entries. For `ApplicationDesktop`, Name is always the application name as provided to `HliInitialize`.

```
Get: (UiObject obj, const char **name)
```

NumColumns cust

AligningForm, Palette:
For an aligning form, tells how many columns are to be used for positioning of the children. Zero is changed to 2.

For a palette, specifies the number of columns in the trays. All trays have the same number of columns.

```
Get:(UiObject obj, HliUnsignedShort* ncol)
Set:(UiObject obj, HliUnsignedShort ncol)
```

OkLabelcust
OkLabelFromMessage
OkLabelFromVMessage

CommandBox :
This label is used for the OK button, used to confirm and dismiss the dialog box. Normally, the application will set null for this attribute, allowing the label to be supplied by Motif or another window environment.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliShortString buf)
Set:(UiObject obj, HliConstShortString buf)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

Orientation

Separator :
Tells whether the separator should be horizontal or vertical.

```
Get:(UiObject obj,  UiOrientation *orient)
Set:(UiObject obj, UiOrientation orient)
```

OtherData

UiObject:
Open space for the application to attach any information it desires to the object instance. Not validated or interpreted in any way by the UI Subsystem.

```
Get:(UiObject obj, void **data)
Set:(UiObject obj, void *data)
```

Parent

BuildingBlock, DialogBox, ApplicationDesktop :
The instance parent. For BuildingBlock, Parent must be set at creation and may only be set at creation. For DialogBox, Parent is always the Application Desktop. For ApplicationDesktop, Parent is always `ui_null_object`.

```
Get:(UiObject obj, UiObject *parent)
```

PercentDrawn

Separator :
Percent of the available spanning distance that should be drawn. For example, if the value is 80, the separator will cover 80% of the distance, leaving 10% blank space at each end.

```
Get:(UiObject obj, HliSignedShort *percentage)
Set:(UiObject obj, HliSignedShort percentage)
```

Pointer

EntryList

PositionInDesktop

BuildingBlock:
The position of the Building Block relative to the Application Desktop, rather than relative to the parent's coordinate space.

```
Get:(UiObject obj,UiDimension *x, UiDimension *y)
```

Prefilled

Field:
Tells whether the field is to have its prefilled value obtained from default, remembered from invocation to invocation, or recomputed (inferred) each time the field is displayed.

```
Get:(UiObject obj, UiPrefilled* prefilled)
Set:(UiObject obj, UiPrefilled prefilled)
```

Prompt
PromptFromMessage
PromptFromVMessage

ApplicationDesktop:
String to appear in the prompt area, to tell the user what to do next. Set by the current tool, depending on the meaning of the next mouse motion.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Set:(UiObject obj, HliConstShortString prompt)
Set...FromMessage (UiObject obj, HliStatus msg, ...)
Set...FromVMessage (UiObject obj, HliStatus msg, va_list args)
```

PromptOn cust

ApplicationDesktop:
A Boolean attribute that indicates whether the prompt area is to be visible.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

PropertiesCb

ToolEntry:
Called when the user asks to edit Tool Properties. This callback should create a Command Argument Box and make it visible, then return immediately, without waiting for it to be completed.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

PushedCb

Toggle, PushButton :
The callback to be invoked when the button is pushed. System Data is null.

```
Get:(UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

RGB

BuildingBlock:
This is the color type of red, green, and blue that can be used as a specifier value.

Syntax for the RGB triple form is red, green, blue. The red, green, and blue values are floating-point values ranging from 0.0 to 1.0 and represent the intensity of the respective color component.

```
Get:(UiObject obj, UiColor color, HliDouble *red, HliDouble
*grn, HliDouble *blu)
Set:(UiObject obj, UiColor color, HliDouble red, HliDouble grn,
HliDouble blu)
```

RadioBehavior

ScrollList :
TRUE means that at most one choice is permitted; when the user has chosen an item, any previously chosen items is not chosen.

```
Get:(UiObject obj, HliBoolean *radBehavior)
Set:(UiObject obj, HliBoolean radBehavior)
```

Remembered

OptionList :
The remembered value. If `Prefilled=ui_remembered`, this value will be copied into TextValue at prefill time.

```
Get:(UiObject obj, HliShortString value)
```

RememberedBoolean

Toggle :
The remembered value from the last time the field was visible and confirmed. If `Prefilled = ui_remembered`, then this value with be copied into Checked at prefill time, for example, every time the parent dialog box becomes visible.

```
Get:(UiObject obj, HliBoolean *value)
```

RememberedItems

ScrollList :
The remembered list of chosen data. An array of items that are to be chosen. If `Prefilled=ui_remembered`, this value will be copied into ChosenItems at prefill time.

```
Get:(UiObject obj, HliSignedShort *rememItems)
```

RememberedItemsNum

ScrollList :
Number of items in RememberedItems.

```
Get:(UiObject obj, HliSignedShort *rememItemsNum)
```

RememberedObject

RadioSet :
This is the remembered choice from the last time the set was visible and confirmed. If `Prefilled=ui_remembered`, then this value normally will be copied into `ChosenObject` at prefill time (if the object is managed, visible, and; otherwise null).

```
Get:(UiObject obj, UiObject *r)
```

RememberedString

Fillin, Paragraph :
The remembered text from the last time the field was visible and confirmed. If `Prefilled = ui_remembered`, then this value will be copied into TextString at prefill time.

```
Get:(UiObject obj, HliString value)
```

RememberedStringNum

Fillin, Paragraph :
Number of characters in the Default String.

```
Get:(UiObject obj, HliSignedShort *num)
```

ResetCb

Pushbutton:
Called when the Reset button is pressed, for example, to initialize the CAB. Before the callback is called, the dialog box contents will be reset. SystemData is unused.

If the CAB has been tacked, the CAB remains tacked even after the Reset button is pressed.

Normally, this callback is left null since the prefilled behavior and initialize callback will handle initialization. The initialize callback is called each time the ResetCb is issued.

```
Get:UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

ResetLabel
ResetLabelFromMessage
ResetLabelFromVMessage

Pushbutton:
The text label for the Reset button.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliShortString buf)
Set:(UiObject obj, HliConstShortString buf)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

RowColumn

BuildingBlock:
Any UiObject that is inside an `AligningForm` has a RowColumn attribute which it takes on. You can use `UiGet` and `UiSet` by specifying the object in the `AligningForm` rather than the Aligning Form itself.

```
Get:(UiObject obj, HliSignedShort* col, HliSignedShort* wide,
HliSignedShort* row, HliSignedShort* high)
Set:(UiObject obj, HliShortString rowLabel, HliSignedShort idx)
```

RowLabel

ScrollMatrix:
One of the RowLabels.

```
Get:(UiObject obj, HliShortString rowLabel, HliSignedShort idx)
Set:(UiObject obj, HliShortString rowLabel, HliSignedShort idx)
```

RowLabels

ScrollMatrix :
An array of HliStrings which make up the labels to be drawn to the left of each row. You can `UiSet` a specific number of labels, but `UiGet` retrieves all of them.

```
Get:(UiObject obj, HliString *rowLabels)
Set:(UiObject obj, HliShortString *rowLabels,HliSignedShort num)
```

RowsHigh

Field:
The number of rows that this field will occupy. This attribute may only be referenced (set or get) if this field is an immediate child of an Aligning Form. Zero or negative numbers are changed to 1.

```
Get:(UiObject obj, HliSignedShort* high)
Set:(UiObject obj, HliSignedShort high)
```

RowsNum

ScrollMatrix:
The number of rows in a `ScrollMatrix` object.

```
Get:(UiObject obj, HliSignedShort *rowsNum)
```

Sensitive cust

UiObject, BuildingBlock:
Whether to display an object normally or to gray it out; to allow user input to it; and to allow focus.

TRUE displays the object normally, allows it to obtain focus, and permits user input (if appropriate for this object). FALSE grays out the object, prevents user input, and disallows focus.

For `BuildingBlock`, even if Sensitive, the object will be portrayed as insensitive if its parent is insensitive.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

SensitiveNow

BuildingBlock:
Tells whether this building block is currently sensitive. True if Sensitive and parent is `SensitiveNow`. Can be true even if the building block is not visible.

```
Get:(UiObject obj, HliBoolean *state)
```

Source cust

UiMenu, Palette:
An attribute passed to the `EntryDefiner` procedure for an entry in a Palette or Menu. Typically, the source attribute specifies the library which should be loaded and initialized. The initialization of the library is expected to create the needed entry or entries.

A value beginning with an @ character indicates that the source is a script file which can be invoked when this entry is activated. if there is an `ActivateCb`, then that callback is performed first, then the script is begun.

```
Get:(UiObject obj, const char ** source)
Set:(UiObject obj, const char * source)
```

Specifier cust

Color, FontCategory:
For `Color`, this is the color specifier string. It can be an RGB triplet, a “crayola” color, or a platform-specific color name.

For `FontCategory`, setting the specifier automatically changes the typeface, size, and style

```
Get:(UiObject obj, UiColor color, const char ** specifier)
Set:(UiObject obj, UiColor color, const char * specifier)
```

StartColumn

Field:
Specifies that a field wishes to start at a particular column location within an Aligning Form parent. (The field must be an immediate child of an Aligning Form.) `ui_unspecified_column` allows the parent to assign any appropriate column, and is presumed if the value specifies more columns than those in the Aligning Form.

```
Get:(UiObject obj, HliSignedShort* col)
Set:(UiObject obj, HliSignedShort col)
```

StartRow

Field:
Specifies that a field wishes to start at a particular row location within an Aligning Form parent. (The field must be an immediate child of an Aligning Form.) `ui_unspecified_row` allows the parent to assign any appropriate row.

```
Get:(UiObject obj, HliSignedShort* row)
Set:(UiObject obj, HliSignedShort row)
```

State

ApplicationState:
Used to tell whether the state is TRUE or FALSE.

```
Get:(UiObject obj, HliBoolean *value)
Set:(UiObject obj, HliBoolean value)
```

StatusArea

DatasetWindow:
Allows status fields to be added to an Aligning Form, which is automatically created when the DatasetWindow is created. This Aligning Form is not necessarily parented in the Dataset Window, so no assumptions can be made about its parentage. The name of the Status Area is the same as that of the Dataset Window, with the letters “Status” appended.

```
Get:(UiObject obj, UiObject *statusArea)
```

Sticky

```
Get:(UiObject obj, UiStickyBits *sticky)
Set:(UiObject obj, UiStickyBits sticky)
```

TextFontCategory

ScrollMatrix :
The font category to be used for the text displayed in cells.

```
Get:(UiObject obj, UiFontCategory *font)
Set:(UiObject obj, UiFontCategory font)
```

TextString
TextStringFromMessage
TextStringFromVMessage

Fillin, Paragraph:
For fillin fields, the current contents of the field. Tabs are permitted, but imbedded line delimiters and other control characters are ignored.

For `paragraphs`, the current contents of the field. Hard line breaks are represented by `hli_line_delimiter`. Tab is also permitted.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliString value)
Set:(UiObject obj, HliConstString value)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

TextStringNum

Fillin, Paragraph :
Number of characters in the `TextString`.

```
Get:(UiObject obj, HliSignedShort *num)
```

TextValue cust
TextValueFromMessage
TextValueFromVMessage

Entry, Label, OptionList, PushButton, Toggle :
For `entries`, `labels`, `pushbuttons` and `toggles`, the string value to be displayed. Strings may have imbedded line delimiters (`hli_line_delimiter`) to show line breaks.

If any character is designated as a mnemonic for a menu `entry`, it must be preceded by an ampersand (&).

For `option lists`, the string value currently chosen. When setting this, ensure that the string is one of the Items.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliShortString value)
Set:(UiObject obj, HliConstShortString value)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

Title cust
TitleFromMessage
TitleFromVMessage

Aligning Form, NonAligning Form:
The title of the form, displayed depending on the platform.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliShortString title)
Set:(UiObject obj, HliConstShortString title)

Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

UserText cust

IdentificationBox:
Text for the user to add through attribute customization.

Visible

BuildingBlock:
Sets the `VisibleWhenManaged` attribute. The outcome is the same as using Set:(UiObject obj, HliBoolean state)

VisibleNow

BuildingBlock:
Whether this building block is visible at the present time. True if the building block is Manage and `VisibleWhenManaged` and the parent is `VisibleNow`. The Application Desktop is defined to be always `VisibleNow`.

```
Get:(UiObject obj, HliBoolean *state)
```

VisibleWhenManaged cust

BuildingBlock, DialogBox, CommandBox :
Display the object. TRUE requires the parent to be visible and Manage to be set.

For `CommandBox`, the box is always created invisible until its children are created, when the box can be made visible.

```
Get:(UiObject obj, HliBoolean *state)
Set:(UiObject obj, HliBoolean state)
```

Width

BuildingBlock:
The current width of the building block, including any border or margin that the building block itself provides, but excluding a border or margin provided externally, for example, by the parent or by the platform's window manager.

Only guaranteed to be correct if VisibleNow.

You can use the convenience functions `UiGetGeometry` or `UiSet Geometry` for the X, Y position, and the width and height.

```
Get:(UiObject obj, UiDimension *width)
```

WidthSpecified

BuildingBlock, DialogBox :
The application's desire for initial box width, exclusive of window decorations, or the dialog box's request for width. The window manager or parent may not honor this preference. If left at default, its value is inferred from the contents' preferred minimum size.

You can use the convenience functions `UiGetGeometrySpecified` or `UiSet GeometrySpecified` for the X, Y position, and the width and height.

```
Get:(UiObject obj, UiDimension *width)
Set:(UiObject obj, UiDimension width)
```

WindowActivateCb

UiCallback:
Called when focus of attention moves from one dataset window to another. User mechanism is platform dependent: For example, user may click in a dataset window to bring it to the front and active, or may use window manager operations. System data is a transient pointer to the UiObject which is the new current active window.

```
Get:UiObject obj, UiCallbackProc* cb, void** userdata)
Set:(UiObject obj, UiCallbackProc cb, void* userdata)
```

WindowTitle cust
WindowTitleFromMessage
WindowTitleFromVMessage

TopLevel, DialogBox, AlignCmdBox, CommandBox:
The title to be displayed in the top of the window.

You can use the `FromMessage` convenience functions to supply a text value from a repository or an Hli string.

```
Get:(UiObject obj, HliString title)
Set:(UiObject obj, HliConstString title)
```

```
Set...FromMessage (UiObject obj, HliStatus message, ...)
Set...FromVMessage (UiObject obj, HliStatus message, va_list args)
```

XPosition

BuildingBlock:
The current horizontal position of the upper left corner of the building block, relative to its parent building block, and including any border or margin provided by the building block but not provided externally, for example, by the parent or by the window manager.

This value is only guaranteed to be correct if VisibleNow. You can use the convenience function `UiGetGeometry` for the X, Y position, and the width and height.

```
Get:(UiObject obj, UiDimension *x)
```

XSpecified

BuildingBlock, DialogBox :
The specification for the horizontal location or the application or dialog box. The parent might not choose to honor this preference.

For `DialogBox`, this value is only used for the first visibility of the box; on following occasions, the previous X position is remembered and overrides the specified values.

You can use the convenience functions `UiGetGeometrySpecified` or `UiSet GeometrySpecified` for the X, Y position, and the width and height.

```
Get:(UiObject obj, UiDimension *x)
Set:(UiObject obj, UiDimension x)
```

YPosition

Buildingblock:
The current vertical position of the upper left corner of the building block, relative to its parent building block, and including any border or margin provided by the building block but not provided externally, for example, by the parent or by the window manager.

This value is only guaranteed to be correct if VisibleNow.

You can use the convenience function `UiGetGeometry` for the X, Y position, and the width and height.

```
Get:(UiObject obj, UiDimension *y)
```

YSpecified

BuildingBlock, DialogBox:
The specification for the vertical location of the application or dialog box. The parent might not choose to honor this preference.

For `DialogBox`, this value is only used for the first visibility of the box; on following occasions, the previous Y position is remembered and overrides the specified values

You can use the convenience functions `UiGetGeometrySpecified` or `UiSet GeometrySpecified` for the X, Y position, and the width and height.

```
Get:(UiObject obj, UiDimension *y)
Set:(UiObject obj, UiDimension y)
```

Chapter 6

CVact Classes

In CVact, a class is the source from which you derive or instantiate objects, whether you use the CVact design tool or the UiCreate functions. Normally, each class provides some attributes and some Ui functions to the objects that can be derived from it.

The class functions section lists the functions and their parameters as they are declared in ANSI C. This means that the data type preceeds the variable name. When the word `const` or the prefix `HliConst` is used to modify the data type, it means that what you pass in is protected from being changed by the internal code. It does not mean that you have to declare the argument you are going to pass in as constant.

This chapter contains information about classes, inheritance hierarchy, relating attributes to objects and uifunctions and attributes that are appropriate to each class.

- Hierarchy of Class Attributes and UI Functions
- ActionEntry
- AligningForm
- ApplicationDesktop
- ApplyPulldown
- CommandBox (CAB)
- CommandEntry
- Container
- DataSetWindow
- DialogBox
- DirectActionEntry

- EditorDialogBox
- Entry
- EntryList
- ErrorBox
- FailureBox
- Fillin
- FloatFillin
- IdentificationBox
- InformationBox
- IntegerFillin
- Label
- NonAligningForm
- OptionList
- Palette
- Paragraph
- PushButton
- QueryResultBox
- QuestionBox
- RadioSet
- ScrollList
- ScrollMatrix
- Separator
- SlideOffEditorDlgBox
- Slider
- SpecifierEntry
- Toggle
- ToggleEntry
- TopLevel
- WarningBox
- WorkingBox

Hierarchy of Class Attributes and UI Functions

The charts on the following pages schematically list the attributes and functions that are available and appropriate to each class of CVact objects.

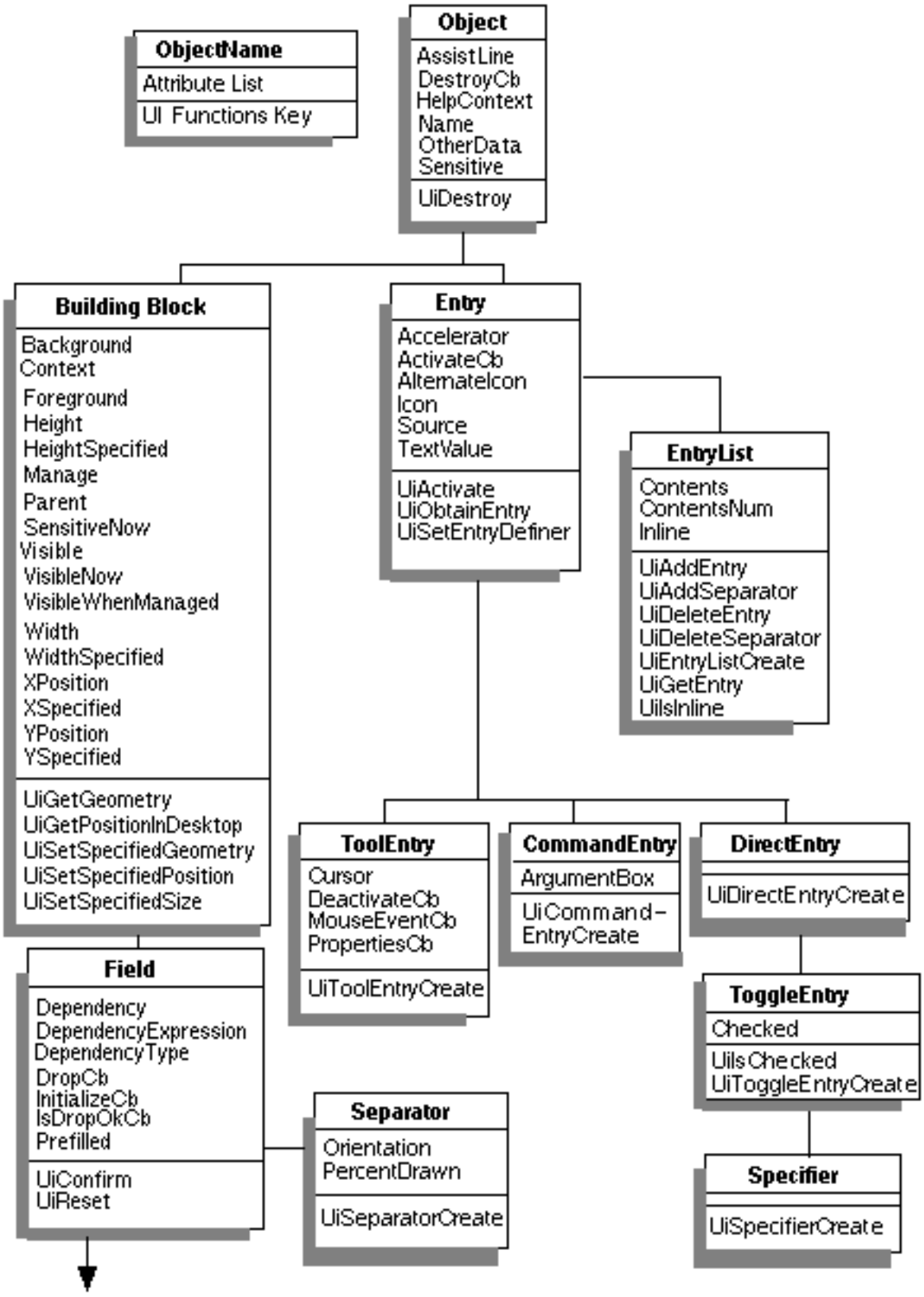
Since CVact is an object-oriented GUI builder, the schema that we present should be thought of as a tree of classes. Each class inherits something from its superclass, although it may not use what it inherits. For example, any object that is instantiated from the class of `AligningForms` possesses an attribute for the number of columns that it takes to display the object. In other words, this column width attribute is inherited by any object that is instantiated from AligningForm or any of its subclasses. This gives the integer and float fillin fields a column width attribute (for example).

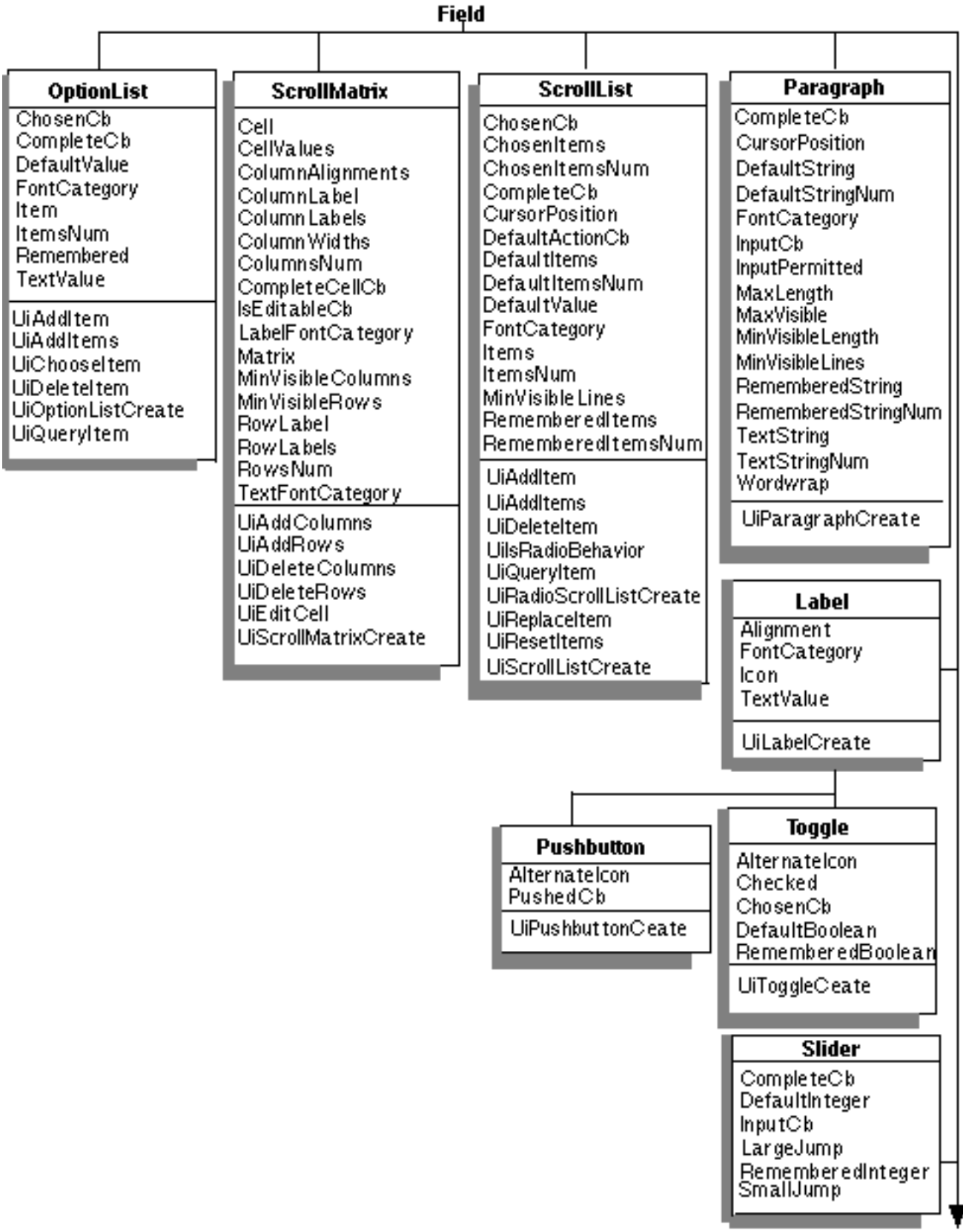
How to Use the Inheritance Tree

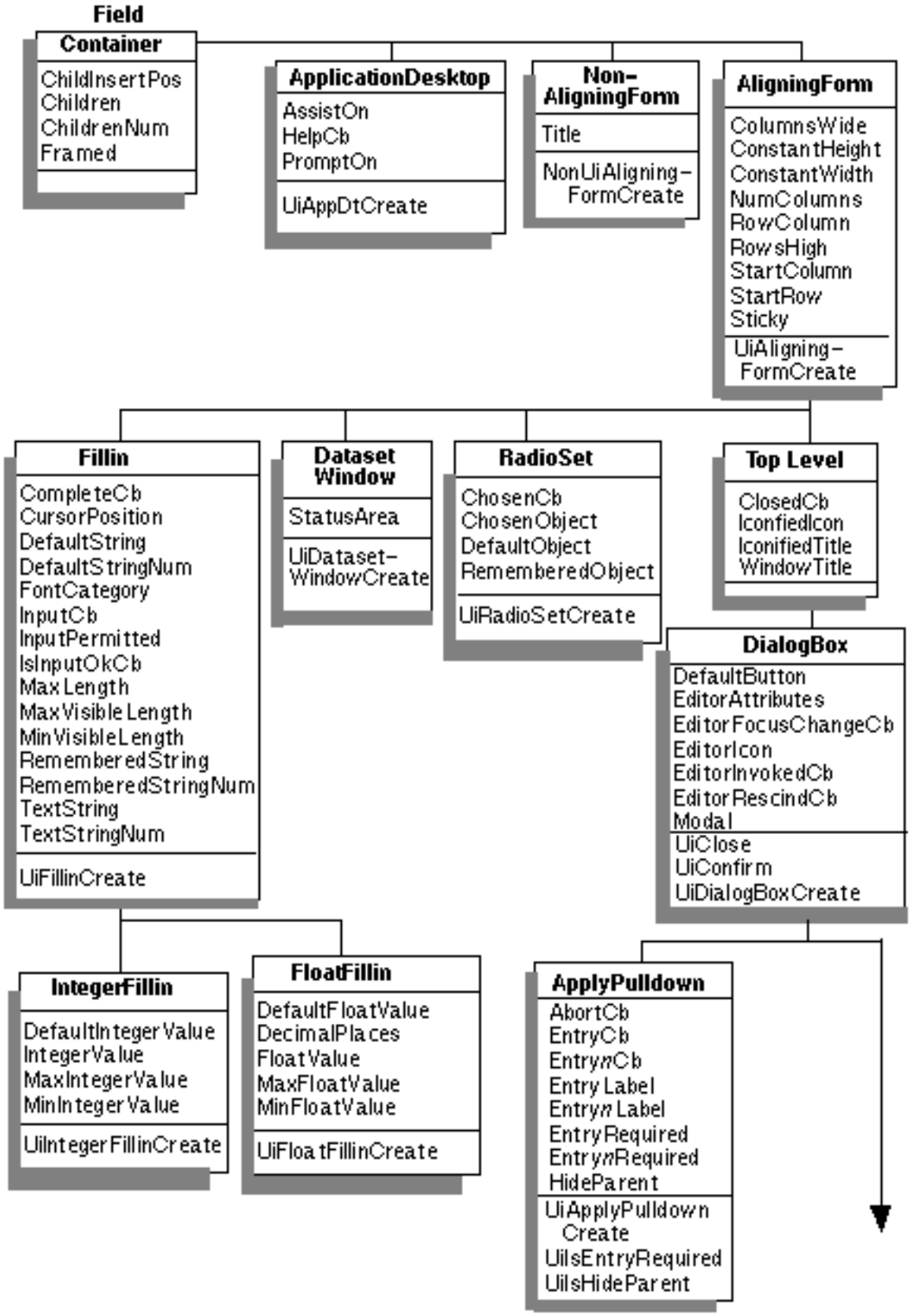
If you want to create an object (using the CVact design tool or your own application), or manipulate the attributes of an object, you need to know what those attributes are. We could list the attributes for each object in the section in this chapter on class descriptions, but it's easier for you to learn how the classes are related to each other, and for you to have a more intuitive understanding of what attributes are available for a given class, and why. You could look in the Properties Editor of the CVact design tool to see which attributes your class supports, but you wouldn't necessarily understand the relevance of these attributes (nor do they all appear in the design tool).

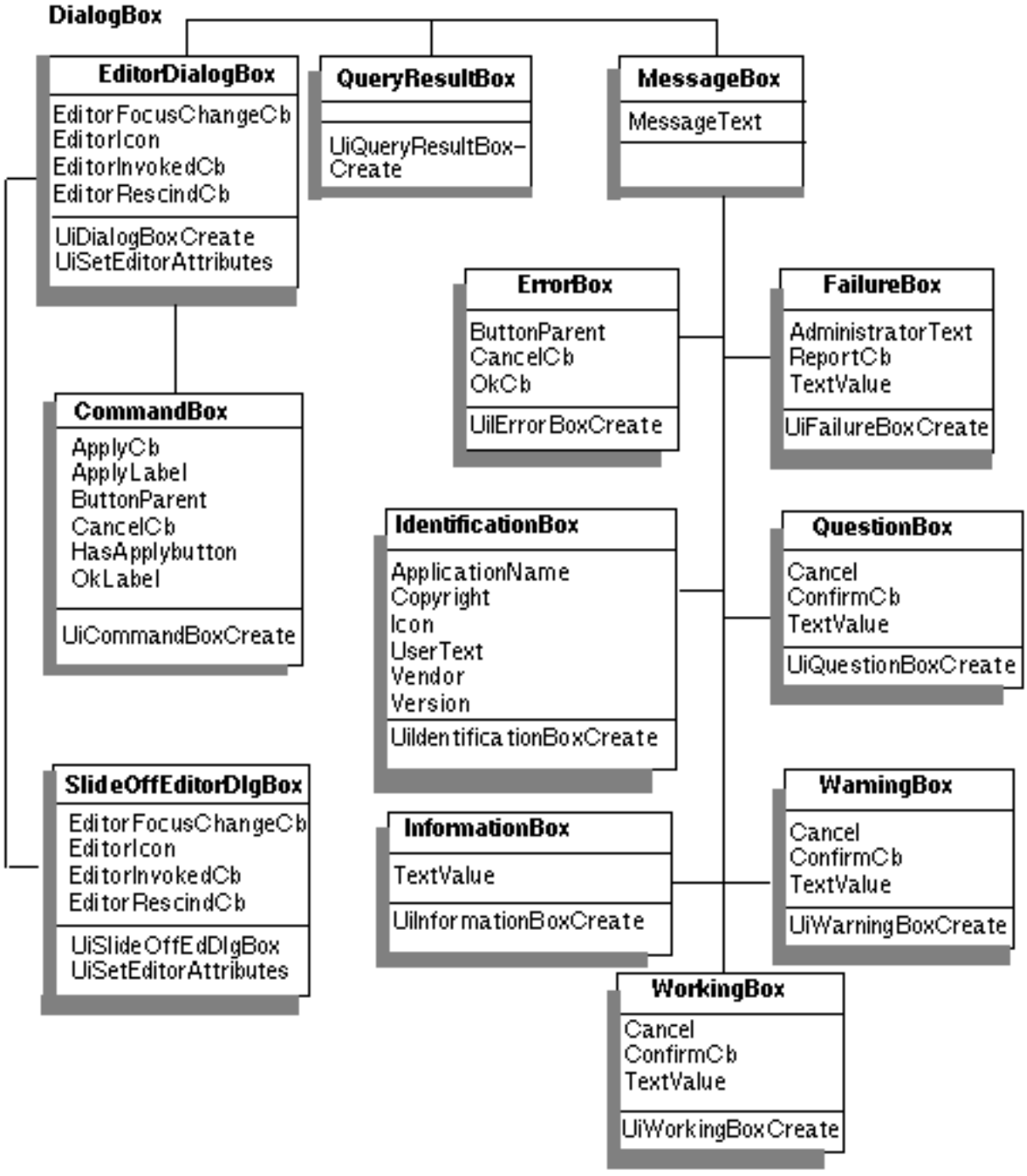
If you are creating an option list, for example, and want to know what attributes and functions it supports:

- Find OptionL in the hierarchy. The attributes and Ui functions in its box are new—no superclass defines them.
- Look in this appendix for OptionList to get a quick definition of these attributes and functions.
- Look in the Attributes appendix for each attribute to get a more detailed description.
- Look in the UiFunctions appendix for details on each function.
- Look up the class tree for attributes and functions that are inherited from Field, BuildingBlock, and Object.


ObjectName
Attribute List
UI Functions Key
Object
AssistLine
DestroyCb
HelpContext
Name
OtherData
Sensitive
UiDestroy
Building Block
Background
Context
Foreground
Height
HeightSpecified
Manage
Parent
SensitiveNow
Visible
VisibleNow
VisibleWhenManaged
Width
WidthSpecified
XPosition
XSpecified
YPosition
YSpecified
UiGetGeometry
UiGetPositionInDesktop
UiSetSpecifiedGeometry
UiSetSpecifiedPosition
UiSetSpecifiedSize
Entry
Accelerator
ActivateCb
AlternateIcon
Icon
Source
TextValue
UiActivate
UiObtainEntry
UiSetEntryDefiner
EntryList
Contents
ContentsNum
Inline
UiAddEntry
UiAddSeparator
UiDeleteEntry
UiDeleteSeparator
UiEntryListCreate
UiGetEntry
UiIsInline
ToolEntry
Cursor
DeactivateCb
MouseEventCb
PropertiesCb
UiToolEntryCreate
CommandEntry
ArgumentBox
UiCommand-
EntryCreate
DirectEntry
UiDirectEntryCreate
Field
Dependency
DependencyExpression
DependencyType
DropCb
InitializeCb
IsDropOkCb
Prefilled
UiConfirm
UiReset
Separator
Orientation
PercentDrawn
UiSeparatorCreate
ToggleEntry
Checked
UiIsChecked
UiToggleEntryCreate
Specifier
UiSpecifierCreate


Field
OptionList
ChosenCb
CompleteCb
DefaultValue
FontCategory
Item
ItemsNum
Remembered
TextValue
UiAddItem
UiAddItems
UiChooseItem
UiDeleteItem
UiOptionListCreate
UiQueryItem
ScrollMatrix
Cell
CellValues
ColumnAlignments
ColumnLabel
ColumnLabels
ColumnWidths
ColumnsNum
CompleteCellCb
IsEditableCb
LabelFontCategory
Matrix
MinVisibleColumns
MinVisibleRows
RowLabel
RowLabels
RowsNum
TextFontCategory
UiAddColumns
UiAddRows
UiDeleteColumns
UiDeleteRows
UiEditCell
UiScrollMatrixCreate
ScrollList
ChosenCb
ChosenItems
ChosenItemsNum
CompleteCb
CursorPosition
DefaultActionCb
DefaultItems
DefaultItemsNum
DefaultValue
FontCategory
Items
ItemsNum
MinVisibleLines
RememberedItems
RememberedItemsNum
UiAddItem
UiAddItems
UiDeleteItem
UiIsRadioBehavior
UiQueryItem
UiRadioScrollListCreate
UiReplaceItem
UiResetItems
UiScrollListCreate
Paragraph
CompleteCb
CursorPosition
DefaultString
DefaultStringNum
FontCategory
InputCb
InputPermitted
MaxLength
MaxVisible
MinVisibleLength
MinVisibleLines
RememberedString
RememberedStringNum
TextString
TextStringNum
Wordwrap
UiParagraphCreate
Label
Alignment
FontCategory
Icon
TextValue
UiLabelCreate
Pushbutton
AlternateIcon
PushedCb
UiPushbuttonCeate
Toggle
AlternateIcon
Checked
ChosenCb
DefaultBoolean
RememberedBoolean
UiToggleCeate
Slider
CompleteCb
DefaultInteger
InputCb
LargeJump
RememberedInteger
SmallJump


Field
Container
ChildInsertPos
Children
ChildrenNum
Framed
ApplicationDesktop
AssistOn
HelpCb
PromptOn
UiAppDtCreate
Non-AligningForm
Title
NonUiAligning-FormCreate
AligningForm
ColumnsWide
ConstantHeight
ConstantWidth
NumColumns
RowColumn
RowsHigh
StartColumn
StartRow
Sticky
UiAligning-FormCreate
Fillin
CompleteCb
CursorPosition
DefaultString
DefaultStringNum
FontCategory
InputCb
InputPermitted
IsInputOkCb
MaxLength
MaxVisibleLength
MinVisibleLength
RememberedString
RememberedStringNum
TextString
TextStringNum
UiFillinCreate
Dataset Window
StatusArea
UiDataset-WindowCreate
RadioSet
ChosenCb
ChosenObject
DefaultObject
RememberedObject
UiRadioSetCreate
Top Level
ClosedCb
IconfiedIcon
IconifiedTitle
WindowTitle
DialogBox
DefaultButton
EditorAttributes
EditorFocusChangeCb
EditorIcon
EditorInvokedCb
EditorRescindCb
Modal
UiClose
UiConfirm
UiDialogBoxCreate
IntegerFillin
DefaultIntegerValue
IntegerValue
MaxIntegerValue
MinIntegerValue
UiIntegerFillinCreate
FloatFillin
DefaultFloatValue
DecimalPlaces
FloatValue
MaxFloatValue
MinFloatValue
UiFloatFillinCreate
ApplyPulldown
AbortCb
EntryCb
EntrynCb
EntryLabel
EntrynLabel
EntryRequired
EntrynRequired
HideParent
UiApplyPulldown Create
UiIsEntryRequired
UiIsHideParent


DialogBox
EditorDialogBox
EditorFocusChangeCb
EditorIcon
EditorInvokedCb
EditorRescindCb
UiDialogBoxCreate
UiSetEditorAttributes
QueryResultBox
UiQueryResultBox-
Create
MessageBox
MessageText
ErrorBox
ButtonParent
CancelCb
OkCb
UilErrorBoxCreate
FailureBox
AdministratorText
ReportCb
TextValue
UiFailureBoxCreate
CommandBox
ApplyCb
ApplyLabel
ButtonParent
CancelCb
HasApplybutton
OkLabel
UiCommandBoxCreate
IdentificationBox
ApplicationName
Copyright
Icon
UserText
Vendor
Version
UiIdentificationBoxCreate
QuestionBox
Cancel
ConfirmCb
TextValue
UiQuestionBoxCreate
SlideOffEditorDlgBox
EditorFocusChangeCb
EditorIcon
EditorInvokedCb
EditorRescindCb
UiSlideOffEdDlgBox
UiSetEditorAttributes
InformationBox
TextValue
UiInformationBoxCreate
WarningBox
Cancel
ConfirmCb
TextValue
UiWarningBoxCreate
WorkingBox
Cancel
ConfirmCb
TextValue
UiWorkingBoxCreate

ActionEntry

Purpose

An ActionEntry activates a menu entry as if the user had selected it. An ActionEntry operates as soon as it is invoked. It may operate on selected data or modify some application state.

User Interactions

An action entry can be referenced inside a menu, specifier bar, or palette tray/drawer. Typically, the TextValue is shown if the entry is in a menu, but if there is no TextValue the Icon is shown. The Icon is shown if the entry is in a palette, but if there is no Icon the TextValue is used. If the entry is in a palette, and has an icon and alternate icon defined, the alternate icon will be displayed when the mouse pointer enters the entry.

Table 6-1 New Class Attributes

Attribute Name	Type	Default Value
Accelerator cust	UiKey	UI_NULL_KEY
ActivateCb	UiCallback	UI_NULL_CALLBACK
AlternateIcon cust	UiIcon	UI_NULL_ICON
Icon cust	UiIcon	UI_NULL_ICON
Source cust	char *	null

Table 6-2 New Functions Defined by this Class

Function	Description
UiActivate ()	Activate a menu entry as if the user had selected it.

AligningForm

Purpose

Used as a container of fields. The aligning form automatically places the fields into rows and columns. Since an aligning form is itself a field, you can put one aligning form inside of another aligning form.

Description

The Aligning Form (AF) performs geometry management on its managed child fields by neatly organizing them into rows and columns. All rows needs not have the same size, nor all columns, unless ConstantWidth or ConstantHeight is TRUE.

Each child can specify a row and/or column that it wishes to start at, or can leave the specification as `UI_UNSPECIFIED_ROW` or `UI_UNSPECIFIED_COLUMN`, in which case the AF will assign a starting row or column. Each child also specifies its number of columns wide and number of rows high. (Rows and columns are numbered from zero. If there are four columns, they are numbered 0,1,2,3.)

Once it is know which rows/columns each child occupies, the optimal width of each column and the optimal height of each row are calculated. (This computation considers the preferred min/max widths/heights of each child, as well as the amount of total space available.) The result is then used to assign to each managed child its x,y position and its width and height. Even invisible children are considered in the computation of row/column sizes, because there must be sufficient space available so that they can be made visible without requiring a recomputation of those sizes.

If there is more space available in a particular position than the field wishes, the stickiness mask is used to position the field within the space.

The AF controls the flow of focus of attention such that children are visited in the proper order (starting at row 0 column 0 and progressing across the row, then to the next row, etc.), and ensures that invisible and insensitive children are skipped over, as are children who are of a display-only class.

Table 6-3 New Class Attributes

Attribute Name	Type	Default Value
Children	UiObject (array)	get-only
ChildrenNum	HliSignedShort	get-only
Framed	HliBoolean	FALSE

Table 6-4 New Functions Defined by this Class

Function	Description
UiAligningFormCreate ()	Create an aligning form.

ApplicationDesktop

Purpose

Serves as the parent for all dataset windows, palettes, dialog boxes, and other top level building blocks. There is only one application desktop.

Description

When the application desktop is created, it automatically causes the menu bar to be created, using the `menuBar.Contents` resource, which describes its contents. Also, it automatically causes the specifier bar to be created, using the `specifierBar.Contents` resource, which describes its content.

In a Motif environment, the application window is not shown on the screen, and each dataset window has its own message area.

Constraints

There may be only one Application Desktop.

The Application Desktop must be the first Building Block created. It does not have to be the first object created.

Table 6-5 New Class Attributes

Attribute Name	Type	Default Value
ActiveWindow	UiObject	get-only
AssistOn cust	HliBoolean	TRUE
CommandOn cust	HliBoolean	FALSE
CurrentTool cust	UiObject	null
HelpCb	UiCallback	UI_NULL_CALLBACK
Prompt	HliString	HLI_NULL_STRING
PromptOn cust	HliBoolean	FALSE
WindowActivateCb	UiCallback	UI_NULL_CALLBACK

Table 6-6 New Functions Defined by this Class

Function	Description
UiApplicationDesktopCreate ()	Create an application desktop.

ApplyPulldown

Purpose

Used to provide an extra level of interaction with the user after he or she has pressed the Apply button in a Command Argument Box (CAB).

Description

This is the dropdown menu displayed over the Apply button in the CAB. It has up to six push buttons, whose labels and actions are controlled by the attributes.

When the user presses any of the six buttons, the appropriate Entry[n]Cb is performed. Presumably, these callbacks will cause text to be sent to the CADDS command line. When it is desired for the ApplyPulldown to be dismissed, the appropriate callback should call either `UiConfirm`, if successfully completed, or `UiClose` if unsuccessful.

Constraints

If you use the CVact interface design tool, you need to put the apply pulldown in a CAB. If you are using a call to `UiApplyPulldownCreate`, the CAB must be the parent object.

Table 6-7 New Class Attributes

Attribute Name	Type	Default Value
AbortCb	UiCallback	UI_NULL_CALLBACK
Entry*n*Cb	UiCallback	UI_NULL_CALLBACK
Entry*n*Label	HliShortString	HLI_NULL_STRING
Entry*n*Required	HliBoolean	FALSE
HelpContext	UiHelpContext	set only at creation
HideParent	HliBoolean	set only at creation
IconifiedIcon	UiIcon	set only at creation
IconifiedTitle	HliShortString	set only at creation
Name	char *	set only at creation
Parent	UiObject	set only at creation
WindowTitle	HliShortString	set only at creation

Table 6-8 New Functions Defined by this Class

Function	Description
UiApplyPulldownCreate ()	Create an apply pulldown.
UiConfirm ()	Confirm all the fields in this pulldown's CAB parent.
UiClose ()	Close the apply pulldown.

CommandBox (CAB)

Purpose

The Command (argument) box is used to get arguments from the user so that a command can be constructed and issued.

Description

The CAB is a specialized dialog box. It contains five built-in push buttons: Ok, Apply, Cancel, Reset, and Help. The text labels for the Ok and Apply buttons are attributes that can be specified. The text labels for the Cancel, Reset, and Help buttons are pre-set, so no attributes are provided to change them.

If you use the CVact interface design tool, you can put additional application-specific fields, such as push buttons, in the form provided. If you use a function call, they can be created by first calling `UiGetButtonParent` to get the parent parameter.

User Interactions

The OK and Apply buttons are insensitive if any visible managed field in the dialog box has the defective attribute set.

When either OK or Apply is pressed, the ApplyCb is performed. If this returns TRUE, then there is an implicit invocation of the “confirm” action on the dialog box, and the dialog box is optionally dismissed, depending on whether the user pressed OK or Apply. When the dialog box is confirmed, all fields perform their `UiConfirm` method, which is important for fields that have the Remembered prefilled attribute.

The Reset button invokes the “reset” action on the dialog box, causing all fields to be reset. The Cancel button just closes.

The application-specific buttons have application-supplied callbacks that determine their meaning.

Constraints

The OkLabel and the ApplyLabel shall be no longer than 20 characters each.

There may be at most three application-defined push button children.

Do not use the CAB as the parent since the CAB is a dialog box and therefore has the same single child constraint as a dialog box.

Table 6-9 New Class Attributes

Attribute Name	Type	Default Value
ApplyCb	UiCallback	UI_NULL_CALLBACK
ApplyLabel cust	HliShortString	HLI_NULL_STRING
ButtonParent	UiObject	get-only
CancelCb	UiCallback	UI_NULL_CALLBACK
HasApplybutton	HliBoolean	set only at creation
OkLabel cust	HliShortString	HLI_NULL_STRING

Table 6-10 Inherited Attributes with New Defaults

Attribute Name	Type	Inherited From	Default Value
TitleString ??	UiCallback	TopLevel	user-invoked command
VisibleWhenManaged cust	HliBoolean	BuildingBlock	false

Table 6-11 New Functions Defined by this Class

Function	Description
UiCommandBoxCreate ()	Create a command argument box.

CommandEntry

Purpose

Display a Command Argument Box (CAB). The CAB prompts the user for additional arguments necessary to complete a command.

Description

User selection of the command entry causes the specified argument box to be activated. If the argument box does not already exist, the command box activate procedure will be invoked to create it.

Table 6-12 New Class Attributes

Attribute Name	Type	Default Value
ArgumentBox	Char *	nameCmdBox

Table 6-13 New Functions Defined by this Class

Function	Description
UiCommandEntryCreate ()	Create a command entry that uses a command argument box.

Container

Purpose

A Container is an object that can have children. This subclass adds attributes and methods pertaining to the parent-child relationship, but since it's an abstract class, you can't make one directly.

Description

The child building blocks are displayed wholly within the space defined by the container for its children. This means that children are displayed within their parent, except when the parent is too small, in which case all or part of the child might be clipped away. When the container is not visible, no child is visible. When the container is insensitive, no child is sensitive.

Constraints

There is at most one child field that is not a push button, and it must be a field.

Table 6-14 New Class Attributes

Attribute Name	Type	Default Value
Children	UiObject (array)	get-only
ChildrenNum	HliSignedShort	get-only
Framed	HliBoolean	FALSE

Table 6-15 New Functions Defined by this Class

Function	Description
UiDestroy ()	Destroy a container and its children.

DataSetWindow

Purpose

Provide a window where the user can work.

Description

This is a window which holds presentation areas. Window decorations are provided so that the window can be moved, resized, iconified, etc. The dataset window has menu bar, palette, message area, prompt area, and so on.

The dataset window is responsible for displaying and managing the direct-manipulation mechanisms by which the user causes panes (presentation areas) to be resized and split.

The parent is the application window, which must have been previously created. Background and foreground color are inherited from the application window.

When a dataset window is first created, it is initially invisible. After appropriate presentation areas and status area contents have been created, the application dialog is expected to make the dataset window visible.

Constraints

All added children must be Presentation Areas.

Table 6-16 New Class Attributes

Attribute Name	Type	Default Value
StatusArea	UiObject	get-only

Table 6-17 New Functions Defined by this Class

Function	Description
UiDatasetWindowCreate ()	Create a dataset window.

DialogBox

Purpose

A Dialog box is suited for displaying an interface that is used often, but does not need to last or remain visible throughout the life of the application.

Description

A dialog box is a top-level window, so it is under control of the window manager.

Constraints

Can have one child at most. The child is usually a form, such as an aligning form, since a form can contain more than one child.

Table 6-18 New Class Attributes

Attribute Name	Type	Object Value
DefaultButton	UiObject	null
EditorFocusChangeCb	UiCallback	UI_NULL_CALLBACK
EditorIcon	UiIcon	UI_NULL_ICON
EditorInvokedCb	UiCallback	UI_NULL_CALLBACK
EditorRescindCb	UiCallback	UI_NULL_CALLBACK
Modal	HliBoolean	TRUE; set only at creation

Table 6-19 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
ClosedCb	UiCallback	HLI_NULL_CALLBACK
HeightSpecified	UiDimension	UI_UNSPECIFIED_DIMENSION
Parent	UiObject	set only at creation
WidthSpecified	UiDimension	UI_UNSPECIFIED_DIMENSION
[X,Y]Specified	UiDimension	UI_UNSPECIFIED_DIMENSION

Table 6-20 Functions Defined by this Class

Function	Description
UiConfirm ()	Confirm all the fields in this dialog box.
UiClose ()	Close the dialog box.
UiDialogBoxCreate ()	Create a dialog box.
UiReset ()	Reset the dialog box.
UiSlideOffDialogBoxCreate ()	Create a dialog box with slide-off behavior.

DirectActionEntry

Purpose

To invoke an immediate action.

Description

Selection of this entry causes the ActivateCb to be invoked. (The ActivateCb is inherited from Entry.)

Table 6-21 Functions Defined by this Class

Function	Description
UiDirectEntryCreate ()	Create a direct action entrydialog box.

EditorDialogBox

Purpose

To help the user fill out or edit the contents of a fillin field. The integer fillin field provides a good example. It comes with a calculator which is an editor dialog box.

Description

This is a specialized dialog box. It is designed to be invoked, or made visible, under control of a fillin field when the user presses an editor icon next to the fillin. Since the fillin field is normally inside another, different, dialog box, it is easiest to visualize the editor dialog box as being a support dialog box for the fillin of the "principal" dialog box.

Associating the EditorDialogBox with a Fillin Field

You can use the CVact interface design tool to associate an editor dialog box with a fillin field, if you supply the box's name, inside double quotes, as the value for the fillin field's Editor attribute. Once you do this, the editor icon button appears next to the fillin.

You can create one within your application by calling `UiCreateEditorDialogBox` and using the normal procedure for assigning an icon to the `UiIcon` parameter (the icon will appear next to the fillin field); then, assuming that the "principal" dialog box has been created and has a fillin field, call `UiSetEditorIcon` (before the principal dialog is set to managed).

Table 6-22 New Class Attributes

Attribute Name	Type	Default Value
EditorFocusChangeCb	UiCallBack	UI_NULL_CALLBACK
EditorIcon	UiIcon	UI_NULL_ICON
EditorInvokedCb	UiCallBack	UI_NULL_CALLBACK
EditorRescindCb	UiCallBack	UI_NULL_CALLBACK

Table 6-23 Functions Defined by this Class

Function	Description
UiEditorDialogBoxCreate ()	Create an editor dialog box.

Entry

Purpose

Used to define the attributes of its derived classes. Because it is an abstract class, it can not be instantiated, but its attributes can be used in (nearly) all inherited objects.

Description

An Entry is not directly user visible. It can be referenced inside a menu, specifier bar, palette tray, or palette drawer. Typically, the TextValue is shown if the entry is in a menu. (The text is rendered using `UI_MENU_Font`.) An entry may be assigned an attribute value that specifies the keyboard accelerator. When this accelerator key is pressed, the effect is as if a related button on the menu or palette was picked. Since the accelerator is a property of the entry, the entry has the same accelerator in all positions where it is referenced.

Mnemonics are specified as part of the TextValue attribute for an entry. Imbedded in the TextValue can be an ampersand (&), preceding the character which is to be the mnemonic. For example, “Select &All” means Select All. The `&` character can be included in the label text by preceding it with an additional `&`, for example., Black & White is displayed as Black & White.

Table 6-24 New Class Attributes

Attribute Name	Type	Default Value
Definer	UiEntryDefinerProc	NULL
DependencyExpression	char *	NULL
TextValue ^{cust}	HliShortString	HLI_NULL_STRING

Table 6-25 Functions Defined by this Class

Function	Description
UiObtainEntry ()	Assign an entry to a menu object.

EntryList

Purpose

A container object used to contain entries and other EntryLists inside menus, EntryLists automate the creation of menus and the menu hierarchy.

Description

If the entry list is in a palette, it's a tray; if it's in a tray, it's a drawer. If it is inside the menu bar, it's a pulldown menu.; if it's in a pulldown menu, it's a cascade menu. An entry list may not be referenced inside a drawer or a cascade menu.

When referenced in a menu, the TextValue of the entry list is displayed. If the entry list is in a palette, the icon displayed is the one associated with the last entry chosen from the drawer (this is the default entry).

An EntryList may have the inline attribute set in the resource file, which means that its entries are brought in inline, or right below the EntryList's TextValue, rather than by creating a cascade menu.

The application dialog does not normally represent the hierarchy of the different kinds of menus. Instead, the hierarchy is represented in resources that are read at the time the application desktop is created or at the time the palette is created. The hierarchy is represented using entry lists that reference entries and other entry lists by name.

Table 6-26 New Class Attributes

Attribute Name	Type	Default Value
Contents cust	array of UiObject	get-only
ContentsNum	HliUnsignedShort	get-only
Inline cust	HliBoolean	set only at creation

Table 6-27 New Functions Defined by this Class

Function	Description
UiAddSeparator ()	Add a separator after a menu entry.
UiAddEntry ()	Add an entry or entry list to a menu.
UiDeleteSeparator ()	Delete a separator in a menu.
UiEntryListCreate ()	Create an EntryList.
UiGetEntry ()	Retrieve an entry or entry list at a specified position in a menu.
UiDeleteEntry ()	Delete an entry or entry list.

ErrorBox

Purpose

Used to inform the user of a user error. Its use is similar to a WarningBox.

Description

An error dialog box is displayed. When the user dismisses the box, the ClosedCb is called.

An error box only displays information containing the TextValue and standard push buttons for the user to acknowledge the error. You could create your own special dialog box for this purpose.

Table 6-28 New Class Attributes

Attribute Name	Type	Default Value
TextValue	HliString	HLI_NULL_STRING

Table 6-29 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Modal	HliBoolean	fixed at TRUE

Table 6-30 New Functions Defined by this Class

Function	Description
UiErrorBoxCreate ()	Creates an error box.

FailureBox

Purpose

Used to inform the user that there is an unrecoverable problem and that the application must shut down.

Description

A failure box provides two messages: one for the user to see, explaining the problem in user terminology, and a second for the system administrator. There are three push buttons:

- Dismiss—Dismisses the dialog box. The ClosedCb is called.
- More Info—Displays additional text intended for the system administrator.
- Generate Report—Calls the reportCb, which should open a file for a bug report and call HliCrash so that a dump or recent history and state can be placed into the bug report.

Because the Modal value is fixed at TRUE, the user will not be able to interact with any other dialog or any other part of the application.

Table 6-31 New Class Attributes

Attribute Name	Type	Default Value
AdministratorText	HliString	HLI_NULL_STRING
ReportCb	UiCallback	UI_NULL_CALLBACK
TextValue	HliString	HLI_NULL_STRING

Table 6-32 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Modal	HliBoolean	fixed at TRUE

Table 6-33 New Functions Defined by this Class

Function	Description
UiFailureBoxCreate ()	Creates a failure box.

Fillin

Purpose

Allow single line text input.

Description

A text fillin field is displayed. If `Editor` is not null, then the editor button (using the icon obtained from the editor dialog box) is displayed and the user can press it, causing the editor to be invoked.

If `InputPermitted` is set, the user may supply text into the field by typing or by pasting from the clipboard.

If not `InputPermitted`, then input may not be supplied, but cursor motion and copy from the field to the clipboard, etc., are permitted.

If the `Defective` attribute is true, the field is visually distinguished. When the Defective attribute is set to TRUE, there may be an audible beep, depending on the platform and/or configuration parameters supplied by the user.

Constraints

The values TextString and DefaultString shall have a length greater than or equal to zero and less than or equal to MaxLength.

MaxLength shall be in the range 1 to 100, inclusive.

Table 6-34 New Class Attributes

Attribute Name	Type	Default Value
CompleteCb	UiCallback	UI_NULL_CALLBACK
CursorPosition	HliSignedShort	0
Defective	HliBoolean	FALSE
DefaultString	HliString	HLI_NULL_STRING
DefaultStringNum	HliSignedShort	get-only
Editor	char *	null
FontCategory	UiFontCategory	UI_TEXT_FIELD_FONT
InputPermitted	HliBoolean	TRUE
IsInputOkCb	UiCallback	UI_NULL_CALLBACK
MaxLength	HliSignedShort	40
MaxVisibleLength	HliSignedShort	40
MinVisibleLength	HliSignedShort	10
RememberedString	HliString	get-only
RememberedStringNum	HliSignedShort	get-only
TextString	HliString	DefaultString
TextStringNum	HliSignedShort	get-only

Table 6-35 New Functions Defined by this Class

Function	Description
UiFillinCreate ()	Creates a text fillin field.

FloatFillin

Purpose

Allow single-line text input of a floating point value.

Description

A text fillin field is displayed with an editor button that leads to a floating-point calculator (Editor Dialog Box). The user may input directly into the text field or use the calculator.

Input is validated on a character-by-character basis to ensure legal format for floating input. An arbitrary number of places to the left and right of the decimal is permitted, as well as an exponent. If negative values aren't permitted (for example., if `MinIntegerValue >=0`), then a minus sign on the mantissa isn't permitted. At field completion, the value is checked to be sure it is legal and within range before the CompleteCb is made. The Defective attribute is set or cleared automatically before the CompleteCb is made, so that it can be interrogated in the callback to see whether the field passed validation.

The display of decimal and thousands separators and exponent are locale-specific, as is the conversion from FloatValue to TextString. However, conversion from TextString to FloatValue is less constrained and will convert values whose meaning is unambiguous.

FloatValue may not be in the range specified by the MinFloatValue and MaxFloatValue, for example, if the Defective attribute were TRUE, or if the user were in the process of inputting values into the field.

Constraints

MinFloatValue <= FloatValue <= MaxFloatValue

Table 6-36 New Class Attributes

Attribute Name	Type	Default Value
DefaultFloatValue	HliDouble	0
DecimalPlaces	HliSignedShort	4
FloatValue	HliDouble	DefaultFloatValue
MaxFloatValue	HliDouble	max (HliDouble)
MinFloatValue	HliDouble	min (HliDouble)

Table 6-37 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Editor	UiObject	UiFloatCalculator
IsInputOkCb	UiCallback	UiFloatCalculator
MaxLength	HliSignedShort	determined by number of digits, thousands and decimal separators, and Max/MinFloatValue
MaxVisibleLength	HliSignedShort	value supplied
MinVisibleLength	HliSignedShort	value supplied

Table 6-38 New Functions Defined by this Class

Function	Description
UiFloatFillinCreate ()	Creates a fillin field for floating point numbers.

IdentificationBox

Purpose

To identify an application (copyright, version, and so on.)

Description

This is a specialized QueryResult box that automatically displays appropriate labels, created from the attributes provided. A Dismiss and Help button are provided.

Table 6-39 New Class Attributes

Attribute Name	Type	Default Value
ApplicationName	HliShortString	HLI_NULL_STRIING
Copyright	HliShortString	HLI_NULL_STRIING
Icon	UiIcon	UI_NULL_ICON
UserText cust	HliShortString	HLI_NULL_STRIING
Vendor	HliShortString	HLI_NULL_STRIING
Version	HliShortString	HLI_NULL_STRIING

Table 6-40 New Functions Defined by this Class

Function	Description
UiIdentificationBoxCreate ()	Creates an identification box.

InformationBox

Purpose

Used to display an information message such as, “Help information can be found by ...” This is a message box like the WarningBox or the ErrorBox, except that its intent is to provide information, rather than notify the user of a problem.

Description

When the user dimisses the box, the Closed Cb is called.

Since the Modal attribute is fixed at false, the user can interact with other dialogs or with the application while the message is being displayed.

Table 6-41 New Class Attributes

Attribute Name	Type	Default Value
TextValue	HliString	HLI_NULL_STRING

Table 6-42 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Modal	HliBoolean	Fixed at FALSE

Table 6-43 New Functions Defined by this Class

Function	Description
UiInformationBoxCreate ()	Creates an information box.

IntegerFillin

Purpose

Allow single-line text input of an integer value. The calculator provided allows Inverse functions of Sin, Cos and Tan.

Description

A text fillin field is displayed with an editor button which leads to a calculator. The user may input directly into the text field using normal fillin field text entry methods or using the calculator.

The calculator is provided with the INV button, which on clicking, sets the calculator to the Inverse mode of operation with which you can perform the Inverse functions of Sin, Cos and Tan. The INV button displays red coloration showing that the Inverse mode of operation is selected. Do not press any other key when the Inverse functions are being performed. If any other key is selected the Inverse mode of operation is aborted which is indicated by the INV button which reverts to its original color.

The calculator is also provided with the SIN, COS and TAN buttons, which on pressing, performs the Sin Inverse, Cos Inverse and Tan Inverse functions respectively. The Numeric entry field displays the values of the Inverse functions. When the Inverse functions are complete, the INV button reverts to its original color from red and the calculator resets the Inverse mode of operation.

Input is validated on a character-by-character basis to ensure that no illegal characters are entered. Also, if negative values aren't permitted (for example., if `MinIntegerValue >=0`), then a minus sign isn't permitted. At field completion, the value is checked to be sure it is legal and within range before the CompleteCb is made. The Defective attribute is also set or cleared automatically before the CompleteCb is made, so the Defective attribute can be interrogated in the callback to see whether the field passed validation.

The display thousands separator is locale-specific, as is the conversion from IntegerValue to TextString. However, conversion from TextString to IntegerValue is less constrained and will convert values whose meaning is unambiguous, even if it doesn't exactly follow the rules of the locale.

The IntegerValue may not be in the range specified by the MinIntegerValue and MaxIntegerValue if, for example, the Defective attribute were TRUE, or the user were in the process of inputting values into the field.

Constraints

MinIntegerValue <= IntegerValue <= MaxIntegerValue

Table 6-44 New Class Attributes

Attribute Name	Type	Default Value
DefaultIntegerValue	HliSignedLong	0
IntegerValue	HliSignedLong	DefaultIntegerValue
MaxIntegerValue	HliSignedLong	max (HliSignedLong)
MinIntegerValue	HliSignedLong	min (HliSignedLong)

Table 6-45 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Editor	UiObject	UiIntCalculator
IsInputOkCb	UiCallback	UiIntCalculator
MaxLength	HliSignedShort	determined by number of digits, thousands separators, and Max/MinIntegerValue
MaxVisibleLength	HliSignedShort	value supplied
MinVisibleLength	HliSignedShort	value supplied

Table 6-46 New Functions Defined by this Class

Function	Description
UiIntegerFillinCreate ()	Creates a fillin field for integers.

Label

Purpose

Display a TextValue and/or Icon.

Description

The TextValue (if any) is displayed together with the Icon (if any). Labels are not available for user input, and may not serve as the focus of attention. If a mnemonic was designated with an & in the TextValue, the designated character will be visually distinguished; if the user gives the mnemonic, the label immediately yields focus to whatever field follows it in focus-of-attention order.

Table 6-47 New Class Attributes

Attribute Name	Type	Default Value
Alignment	UiAlignment	UI_NULL_ICON
FontCategory	UiFontCategory	null
Icon	UiIcon	UI_NULL_CALLBACK
TextValue	HliShortString	UI_NULL_CALLBACK

Table 6-48 New Functions Defined by this Class

Function	Description
UiLabelCreate ()	Creates a label for an object.

NonAligningForm

Purpose

A NonAligningForm (NAF) is used when visual alignment into rows and columns is not desired.

Description

The nonaligning form allows each child field to determine its own x,y position. In most cases, this comes from the attributes XSpecified and YSpecified.

The nonaligning form controls the flow of focus of attention such that children are visited in the order of their x,y position, traversing horizontally among all fields that are at the same y position, then going down to children of the next lower y position.

The preferred height and width of the nonaligning form are taken from the specified values, if provided, otherwise they are computed from the sizes of the child fields.

Constraints

All children shall be of class Field.

Table 6-49 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Framed	HliBoolean	FALSE (fixed)

Table 6-50 New Functions Defined by this Class

Function	Description
UiNonAligningFormCreate ()	Creates a non-aligning form.

OptionList

Purpose

Present non-scrollable menu items for selection.

Description

An option menu is displayed as a button showing a value. When the button is pressed, a drop-down list of choices is provided, and the user may choose one, which becomes the new value and is shown in the button. There is always one item chosen, and whenever a new item is chosen, the previous item is implicitly not chosen.

Please note: Use option lists for lists with 10 items or fewer. Use scroll lists instead of option lists for lists with more than 10 items.

Table 6-51 New Class Attributes

Attribute Name	Type	Default Value
ChosenCb	UiCallback	UI_NULL_CALLBACK
CompleteCb	UiCallback	UI_NULL_CALLBACK
DefaultValue	HliShortString	HLI_NULL_STRING
FontCategory	UiFontCategory	UI_MINOR_TEXT_FIELD_FONT
ItemsNum	HliSignedShort	get-only
Remembered	HliShortString	get-only
TextValue	HliShortString	HLI_NULL_STRING

Table 6-52 New Functions Defined by this Class

Function	Description
UiOptionListCreate ()	Creates an option list.
UiAddItem ()	Adds an item to an option list.
UiAddItems ()	Adds multiple items to an option list.
UiDeleteItem ()	Deletes an item from the list.
UiGetItem ()	Gets the text of an item from a position in the list
UiQueryItem ()	Finds the position of an item by its text value; also returns its chosen state.

Palette

Purpose

Used as a row column container for graphic or textual Menu entries, palettes are created and managed automatically by the contents described in EntryLists.

Description

The contents are described in an entry list named `name-contents`. For example, if the palette is named detailing, the entry list is named `detailing-contents`.

The number of rows in the palette is the largest of the number of rows in each tray. The number of columns in the palette is as specified in the attribute.

Table 6-53 New Class Attributes

Attribute Name	Type	Default Value
NumColumns cust	HliUnsignedShort	2

Table 6-54 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Parent	UiObject	The application desktop.
Title	HliString	TextValue of the entry list defining the palette contents.

Table 6-55 New Functions Defined by this Class

Function	Description
UiPaletteCreate ()	Create an icon palette.

Paragraph

Purpose

Allows scrolling of the multi-line text by means of the scrollbars and entry of wordwrapped multi-line text input.

Description

If InputPermitted is set, the user may supply text into the field by typing, pasting from the clipboard, etc.

If not InputPermitted, then input may not be supplied, but cursor motion and copy from the field to the clipboard, etc., are permitted.

Vertical and horizontal scrollbars are provided for scrolling text upwards and downwards and sideways. You can switch the scrollbars ON and OFF. You can also scroll text using keyboard arrow keys.

A Text widget is created inside a Scrolled window which is a Motif XmScrolledWindow widget which provides a Scrollable window. Earlier this class created a Text widget which is a Motif XmText widget.

Constraints

The values TextString and DefaultString shall have a length greater than or equal to zero and less than or equal to MaxLength.

Table 6-56 New Class Attributes

Attribute Name	Type	Default Value
CompleteCb	UiCallback	UI_NULL_CALLBACK
CursorPosition	HliSignedShort	0
DefaultString	HliString	HLI_NULL_STRING
DefaultStringNum	HliSignedShort	get-only
FontCategory	UiFontCategory	UI_PARAGRAPH_FONT
InputCb	UiCallback	UI_NULL_CALLBACK
InputPermitted	HliBoolean	TRUE
MaxLength	HliSignedShort	1,000
MaxVisibleLength	HliSignedShort	40
MaxVisibleLines	HliSignedShort	
MinVisibleLength	HliSignedShort	40
MinVisibleLines	HliSignedShort	5
RememberedString	HliString	get-only
RememberedStringNum	HliSignedShort	get-only
TextString	HliString	HLI_NULL_STRING
TextStringNum	HliSignedShort	get-only
WordWrap	HliBoolean	TRUE

Table 6-57 New Functions Defined by this Class

Function	Description
UiParagraphCreate ()	Creates a paragraph.

PushButton

Purpose

Provide an easy input method, usually to specify an action.

Push buttons can display text or graphics. If displaying graphics, they can display an icon and an alternate icon when the button is pushed.

Push buttons are inherited from Labels, and as labels they can be set to sensitive, insensitive, visible, invisible and so on.

Description

The behavior of the push button is straight forward. The PushedCb is called when the user places the cursor on the push button and presses and releases the left mouse button.

If the text value of a push button contains a mnemonic (that is, a letter preceded by an `&`), user input of the mnemonic is treated as a press of the key.

Constraints

Although a push button can be parented directly by a dialog box, it wouldn't be practical: a dialog box is allowed only one child, so the push button would be the only thing it contained. In practice, push buttons are made children of container objects such as aligning or non aligning forms. Inside a CAB, application-defined push buttons must be parented by such container objects.

Usage Tips

If push buttons are used to start a lengthy routine, the routine should make frequent calls to the `UiHandlePendingEvents` routine. It may also be advisable to make the push button that activated the lengthy call insensitive in order to prevent multiple unwanted events from being placed on the event queue.

Table 6-58 New Class Attributes

Attribute Name	Type	Default Value
AlternateIcon	UiIcon	UI_DEFAULT_ICON
PushedCb	UiCallback	UI_NULL_CALLBACK

Table 6-59 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Alignment	UiAlignment	fixed at UI_CENTER_ALIGN

Table 6-60 New Functions Defined by this Class

Function	Description
UiPushButtonCreate ()	Creates a pushbutton.

QueryResultBox

Purpose

Displays information which is the result of a query.

Description

The user can dismiss the box when finished examining the information, or can ask for help. Otherwise, no input is expected. This is a general dialog box which provides two buttons: Dismiss and Help.

As in all dialog boxes, there is only one child field, presumably an aligning form. Typically, your application will put in display-only fields or fillin fields for which InputPermitted=FALSE. The intention is that the results of a Query are for display only, but fillin fields are permitted so that the user can copy its contents to the clipboard.

Table 6-61 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
VisibleWhenManaged	HliBoolean	FALSE The box is made visible only after the children are added.

Table 6-62 New Functions Defined by this Class

Function	Description
UiQueryResultBoxCreate ()	Creates a query result box.

QuestionBox

Purpose

Used to ask the user a “yes” or “no” question.

Description

A question dialog box is displayed. When the user answers yes, the ConfirmCb is called. For no, the CancelCb is called. In either case, the dialog box is then dismissed.

Table 6-63 New Class Attributes

Attribute Name	Type	Default Value
CancelCb	UiCallback	UI_NULL_CALLBACK
ConfirmCb	UiCallback	UI_NULL_CALLBACK
TextValue	HliShortString	HLI_NULL_STRING

Table 6-64 New Functions Defined by this Class

Function	Description
UiQuestionBoxCreate ()	Creates a question box.

RadioSet

Purpose

Used to contain toggle buttons in situations where only one selection is permissible (named for the buttons on a car radio). Can also contain labels and separators.

Description

Displays a set of toggles so that at most one of the objects is the chosen one at all times. Insensitive, invisible, or unmanaged children cannot be chosen.

Table 6-65 New Class Attributes

Attribute Name	Type	Default Value
ChosenCb	UiCallback	UI_NULL_CALLBACK
ChosenObject	UiObject	null
DefaultName [cust]	HliShortString	HLI_NULL_STRING
RememberedObject	UiObject	get-only

Table 6-66 New Functions Defined by this Class

Function	Description
UiRadioSetCreate ()	Creates a set of radio buttons.

ScrollList

Purpose

Present scrollable, menu-like entries for selection.

Description

A list of text items is displayed. The user may scroll vertically and horizontally to see any hidden areas. Items may be chosen or not chosen using the mouse. If RadioBehavior is set, then at most one item may be chosen.

Please note: Use scroll lists for lists with more than 10 items. Use option lists instead of scroll lists for lists with 10 items or less.

Table 6-67 New Class Attributes

Attribute Name	Type	Default Value
ChosenCb	UiCallback	UI_NULL_CALLBACK
ChosenItems	HliSignedShort [ ]	get-only
ChosenItemsNum	HliSignedShort	get-only
CompleteCb	UiCallback	UI_NULL_CALLBACK
CursorPosition	HliSignedShort	0
DefaultActionCb	UiCallback	UI_NULL_CALLBACK
DefaultItems	HliSignedShort [ ]	empty
DefaultItemsNum	HliSignedShort	get-only
DefaultValue	HliShortString	set only at creation
MinVisibleLines	HliSignedShort	5
RadioBehavior	HliBoolean	FALSE
RememberedItems	HliSignedShort [ ]	get-only
RememberedItemsNum	HliSignedShort	get-only

Table 6-68 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
InitializeCb	UiCallback	see *Attributes*, p.
Prefilled	UiPrefilled	UI_INFERRED

Table 6-69 New Functions Defined by this Class

Function	Description
UiAddItem ()	Adds an item at a given position.
UiAddItems ()	Adds multiple items at a given position.
UiDeleteItem ()	Deletes an item at a given position.
UiGetItem ()	Gets the text value of an item.
UiQueryItem ()	Finds position and state of an item corresponding to a text value.
UiRadioScrollListCreate ()	A convenience function to create a scroll list with radio behavior.
UiSetChosen ()	Sets/resets an item's chosen value.
UiScrollListCreate ()	Creates scroll list.

ScrollMatrix

Purpose

Present a two-dimensional array of text strings for editing.

Description

Only one cell in the matrix at a time may receive input focus, and it is this cell which is editable. The matrix may be scrolled horizontally and vertically.

Table 6-70 New Class Attributes

Attribute Name	Type	Default Value
CellColorValues	*UiColor [ ]	null
CellValues	*HliString [ ]	null; set-only
ColumnAlignments	UiAlignment [ ]	UI_START_ALIGN
ColumnLabels	HliShortString [ ]	null
ColumnWidths	HliSignedShort [ ]	10
ColumnsNum	HliSignedShort	set only at creation
CompleteCellCb	UiCallback	UI_NULL_CALLBACK
IsEditableCb	UiCallback	UI_NULL_CALLBACK
IsInputOkCb	UiCallback	UI_NULL_CALLBACK
LabelFontCategory	UiFontCategory	UI_MAJOR_LABEL_FONT
MinVisibleColumns	HliSignedShort	1
MinVisibleRows	HliSignedShort	1
RowLabels	HliShortString [ ]	null
SelectCellCb	UiCallback	UI_NULL_CALLBACK
SelectedColumns	HliSignedShort [ ]	null
SelectedColumnsNum	HliSignedShort	0
SelectedRows	HliSignedShort [ ]	null
SelectedRowsNum	HliSignedShort	0
TextFontCategory	UiFontCategory	UI_TEXT_FIELD_FONT
TopRowNum	HliSignedShort	0

Table 6-71 New Functions Defined by this Class

Function	Description
UiAddColumns ()	Adds columns to a scroll matrix.
UiAddRows ()	Adds rows to a scroll matrix.
UiDeleteColumns ()	Deletes columns from a scroll matrix.
UiDeleteRows ()	Deletes rows from a scroll matrix.
UiEditCell ()	Displays a particular cell from a scroll matrix so that the user can edit it.
UiGetCell ()	Gets the value of a cell.
UiGetCellColor ()	Gets the color value of a cell.
UiGetColumnColors ()	Gets the color values of a column.
UiGetRowColors ()	SelectedRows
SelectedRows	HliSignedShort [ ]
SelectedRowsNum	HliSignedShort
TextFontCategory	UiFontCategory
TopRowNum	HliSignedShort
SelectedRows	HliSignedShort [ ]
SelectedRowsNum	HliSignedShort
TextFontCategory	UiFontCategory
TopRowNum	HliSignedShort
SelectedRows	HliSignedShort [ ]

Separator

Purpose

Visually separate fields in a display.

Description

A line is drawn spanning the parent field. If placed into an aligning form, the line spans the number of columns designated with the ColumnsWide attribute. No user interaction is permitted.

Constraints

PercentDrawn can be between 0 and 100 inclusive.

Table 6-72 New Class Attributes

Attribute Name	Type	Default Value
Orientation	UiOrientation	UI_HORIZONTAL
PercentDrawn	HliSignedShort	80

Table 6-73 New Functions Defined by this Class

Function	Description
UiSeparatorCreate ()	Creates a separator.

SlideOffEditorDlgBox

Purpose

Helps you fill out or edit the contents of a fillin field. The integer fillin field provides a good example. The SlideOffEditorDlgBox exhibits a slide off behavior. It is useful for interactive designing.

Description

This is a specialized dialog box. It is designed to be invoked, or made visible, under control of a fillin field when you press an editor icon next to the fillin. Since the fillin field is normally inside another, different, dialog box, it is easy to visualize the editor dialog box as being a support dialog box for the fillin of the "principal" dialog box. When you slide the cursor out of the slideoffeditordialog, the slideoffeditordialogbox is not visible any more.

The SlideOffEditorDlgBox can be created from the palette using the drag and drop procedure.

Associating the SlideOffEditorDlgBox with a Fillin Field

You can use the CVact interface design tool to associate a slideoff editor dialog box with a fillin field, if you supply the box's name, inside double quotes, as the value for the fillin field's Editor attribute. Once you do this, the editor icon button appears next to the fillin.

You can create one within your application by calling `UiSlideOffDialogBoxCreate` and using the normal procedure for assigning an icon to the `UiIcon` parameter (the icon will appear next to the fillin field); then, assuming that the "principal" dialog box has been created and has a fillin field, call `UiSetEditorIcon` (before the principal dialog is set to managed).

Table 6-74 New Class Attributes

Attribute Name	Type	Default Value
EditorFocusChangeCb	UiCallBack	UI_NULL_CALLBACK
EditorIcon	UiIcon	UI_NULL_ICON
EditorInvokedCb	UiCallBack	UI_NULL_CALLBACK
EditorRescindCb	UiCallBack	UI_NULL_CALLBACK

Table 6-75 Functions Defined by this Class

Function	Description
UiSlideOffDlgBoxCreate ()	Creates a slide off editor dialog box.

Please note: Do not use a calculator within a slideoff editor dialog box. The calculator itself uses a slideoff editor dialog box. You should not use a slideoff editor dialog box within a slideoff editor dialog box.

If no title is specified for the title bar, the default title “TitleBar” is not visible. It does not take the default option for the title as “Title Bar”.

Slider

Purpose

Provides you the facility to enter a value by a slide operation with direct manipulation of the mouse. Also enables both Integer and Float readout through a scale. Allows you to specify the number of decimal places in case of Float readout. However it can only be used programmatically for interactive design.

Description

It is an analog numeric display/input device. You can change the value by direct manipulation using the mouse, for example, moving the position of the slider.

You complete a slide operation when you release the slider. On some platforms, a special mouse click is used to increment or decrement the value of a slider by a fixed large or small amount respectively. This mechanism is used instead of dragging.

This object class is created from the palette using the drag and drop mechanism.

Constraints

MinIntegerValue <= SmallJump <= MaxIntegerValue

MinIntegerValue <= LargeJump <= MaxIntegerValue

Table 6-76 New Class Attributes

Attribute Name	Type	Default Value
CompleteCb	UiCallback	UI_NULL_CALLBACK
DecimalPlaces	Int	0
DefaultInteger	HliSignedLong	0
InputCb	UiCallBack	UI_NULL_CALLBACK
LargeJump	HliSignedLong	10
RememberedInteger	HliSignedLong	get-only
SmallJump	HliSignedLong	1

Table 6-77 New Functions Defined by this Class

Function	Description
UiSliderCreate ()	Creates a slider.

SpecifierEntry

Purpose

Used to display a specifier in the specifier bar.

Description

Table 6-78 New Functions Defined by this Class

Function	Description
UiSpecifierCreate ()	Creates a specifier.

Toggle

Purpose

Present a two-state button that the user can toggle.

Description

This is presented as a check-mark or other indication that changes when the user clicks on it, so that it toggles from a checked to an non-checked state. Whenever the user changes the value, the Chosen callback is performed.

To make a choose-many (multiple-choice) set, place a number of these into an aligning form that is one column wide and has a frame and title. If radio behavior is desired, use a radio set instead.

Table 6-79 New Class Attributes

Attribute Name	Type	Default Value
AlternateIcon	UiIcon	UI_NULL_ICON
Checked	HliBoolean	FALSE
ChosenCb	UiCallBack	UI_NULL_CALLBACK
DefaultBoolean	HliBoolean	FALSE
RememberedBoolean	HliBoolean	get-only

Table 6-80 New Functions Defined by this Class

Function	Description
UiToggleCreate ()	Creates a toggle button.

ToggleEntry

Purpose

Used as a special case of direct action entries that set a Boolean state in the application.

Description

Selection of this entry causes the state of the toggle to change.

Table 6-81 New Class Attributes

Attribute Name	Type	Default Value
Checked cust	HliBoolean	FALSE

Table 6-82 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
ActivateCb	UiCallback	UI_NULL_CALLBACK

Table 6-83 New Functions Defined by this Class

Function	Description
UiToggleEntryCreate ()	Creates a toggle entry.

TopLevel

Purpose

A TopLevel building block is one that is under control of the window manager. This subclass therefore adds behavior related to being iconized, closed, and so on.

Description

The platform's window manager provides window decorations that allow the window to be moved, resized, iconified, and so on.

Table 6-84 New Class Attributes

Attribute Name	Type	Default Value
ClosedCb	HliCallback	HLI_NULL_CALLBACK
IconifiedIcon	UiIcon	UI_NULL_ICON
IconifiedTitle	HliShortString	HLI_NULL_STRING
WindowTitle	HliShortString	HLI_NULL_STRING

WarningBox

Purpose

To display warning messages to the user, such as, “Are you sure you want to delete this part?”

Description

A warning dialog box is displayed with a message in its own dialog box in such a way that the user must give a response. When the user confirms or cancels, the dialog box is dismissed.

Table 6-85 New Class Attributes

Attribute Name	Type	Default Value
Cancel	UiCallback	UI_NULL_CALLBACK
ConfirmCb	UiCallback	UI_NULL_CALLBACK
Text Value	HliString	HLI_NULL_STRING

Table 6-86 Inherited Attributes with New Defaults

Attribute Name	Type	Default Value
Modal	HliBoolean	fixed at TRUE

Table 6-87 New Functions Defined by this Class

Function	Description
UiWarningBoxCreate ()	Creates a warning box.

WorkingBox

Purpose

Used to show that a task or operation is in progress.

Description

A Working dialog box, in the platform-specific format is displayed. If the user Cancels the dialog box, a callback is made and the application should terminate the operation that was in-progress.

Typically, the application will provide a display-only field as a child of this dialog box, to show progress to the user. This field should be a Thermometer (to show progress toward a known completion, or percent complete), or a Label, periodically updated to show stages of completion.

The application must remember to UiClose this dialog box when the work is complete, because the user is unable to do anything while this box is up. For the same reason, this box should not be used if the user is permitted to interact with the application while the application is working (for example, if the box is being painted).

Table 6-88 New Class Attributes

Attribute Name	Type	Default Value
TextValue	HliShortString	HLI_NULL_STRING
ConfirmCb	UiCallback	UI_NULL_CALLBACK
CancelCb	UiCallback	UI_NULL_CALLBACK

Table 6-89 New Functions Defined by this Class

Function	Description
UiWorkingBoxCreate ()	Creates a working box.

Appendix A

Ui Functions

The Ui functions were designed almost exclusively for building general user interfaces, but they have generally been restricted to building just those interfaces that are relevant to CADDS and the company's applications. In other words, they build an idiosyncratic interface. We have taken a general-purpose GUI builder and have simplified it and the objects that it creates.

This appendix lists the Ui functions provided by CVact.

- Attributes
- UiActivate
- UiAddColumns
- UiAddEntry
- UiAddItem
- UiAddItems
- UiAddRows
- UiAddSeparator
- UiAligningFormCreate
- UiAppDtCreate
- UiApplicationStateCreate
- UiApplyPulldownCreate
- UiAssistOn
- UiChooseItem
- UiClose
- UiCommandBoxCreate
- UiCommandEntryCreate

- UiCompletion
- UiConfirm
- Ui...CreateFromVStatus
- UiDatasetWindowCreate
- UiDeleteColumns
- UiDeleteEntry
- UiDeleteItem
- UiDeleteRows
- UiDeleteSeparator
- UiDestroy
- UiDialogBoxCreate
- UiDirectEntryCreate
- UiEditCell
- UiEditorDialogBoxCreate
- UiEndBusy
- UiEntryListCreate
- UiErrorBoxCreate UiErrorBoxCreateFromStatus UiErrorBoxCreateFromVStatus
- UiFailureBoxCreate
- UiFillinCreate
- UiFillinStrlen
- UiFloatFillinCreate
- UiGet...
- UiGet...Cb
- UiGetCell
- UiGetCellColor
- UiGetColumnColors
- UiGetColumnLabel
- UiGetDependency
- UiGetEntry
- UiGetEntryLabel
- UiGetEntryRequired
- UiGetFocus

- UiGetGeometry
- UiGetGeometrySpecified
- UiGetInstanceId
- UiGetItem
- UiGetItemSensitive
- UiGetObjectByName
- UiGetPositionInDesktop
- UiGetResetInProgress
- UiGetRowColors
- UiGetRowColumn
- UiGetRowLabel
- UiGetRowLabels
- UiIdentificationBoxCreate
- UiInformationBoxCreate UiInformationBoxCreateFromStatus UiInformationBoxCreateFromVStatus
- UiIntegerFillinCreate
- UiIsDefaultButton
- UiLabelCreate
- UiNonAligningFormCreate
- UiObtainEntry
- UiOptionListCreate
- UiPaletteCreate
- UiParagraphCreate
- UiParagraphInsertText
- UiParagraphMoveCursor
- UiPostCab
- UiPromptOn
- UiPushButtonCreate
- UiQueryItem
- UiQueryItmStatusByIdx
- UiQueryResultBoxCreate
- UiQuestionBoxCreate UiQuestionBoxCreateFromStatus UiQuestionBoxCreateFromVStatus

- UiRadioScrollListCreate
- UiRadioSetCreate
- UiReset
- UiResetCab
- UiScrollListCreate
- UiScrollMatrixCreate
- UiSeparatorCreate
- UiSet...
- UiSetBorderWidthForSldoffDlgBox
- UiSet...Cb
- UiSetCell
- UiSetCellColor
- UiSetCellValues
- UiSetChosen
- UiSetColumnAlignments
- UiSetColumnColors
- UiSetColumnLabel
- UiSetColumnLabels
- UiSetDefaultButtonWithSThickness
- UiSetDependency
- UiSetEditorAttributes
- UiSetEntryLabel
- UiSetEntryRequired
- UiSet...FromMessage UiSet...FromVMessage
- UiSetGeometrySpecified
- UiSetInstanceId
- UiSetItemSensitive
- UiSetMatrix
- UiSetRowColor
- UiSetRowColumn
- UiSetRowLabel
- UiSetRowLabels

- UiSlideOffDialogBoxCreate
- UiSlideOffEditorDialogBoxCreate
- UiSetSpecifiedPosition
- UiSetSpecifiedSize
- UiSpecifierCreate
- UiStartBusy
- UiToggleCreate
- UiToggleEntryCreate
- UiWarningBoxCreate UiWarningBoxCreateFromStatus UiWarningBoxCreateFromVStatus
- UiWorkingBoxCreate

Attributes

Many of the Ui functions are concerned with getting and setting attributes. Often, if;

- you know the attribute name, you can construct the appropriate `UiGet` or `UiSet` call—see `UiGet...` for an example.
- the attribute is a callback, you can also construct the proper call—see `UiGet...Cb` for an example.

We list all of the attributes here, except for callbacks; for these, see `UiGet...Cb` or `UiSet...Cb`. The table uses these conventions:

- The superscript ^{cust} indicates an attribute that can be customized by the user.
- A blank cell in the Get or Set column means that the attribute can't be retrieved or set, respectively.
- The note see desc indicates that the attribute doesn't follow the rules for getting and setting most attributes. See the detailed description found later in the appendix.

Table A-1 Attribute Names and TypeSets

Attribute Name	Get	Set
Accelerator	UiKey*	UiKey
ActiveWindow		
Alignment	UiAlignment *	UiAlignment
AlternateIcon^{cust}	UiIcon *	UiIcon
ApplicationDesktop		
ApplyButton	HliBoolean *	
ApplyLabel^{cust}	HliString	HliConstShortString
ArgumentBox	const char **	const char *
AssistLine	HliShortString	HliConstShortString
AssistOn	see desc	see desc
Background	UiColor *	UiColor
Button	UiObject *	
ButtonParent	UiObject *	
CancelLabel	HliShortString	HliConstShortString
Cell	see desc	see desc
CellColor	see desc	see desc
CellValues		see desc
Checked	HliBoolean *	HliBoolean
Children	UiObject [ ]	

Table A-1 Attribute Names and TypeSets

Attribute Name	Get	Set
ChildrenNum	HliSignedShort *	
Chosen		see desc
ChosenItems	HliSignedShort *	
ChosenItemsNum	HliSignedShort *	
ChosenObject	UiObject *	UiObject
Class	UiClsTypePtr	
ColumnAlignments	UiAlignment *	see desc
ColumnColors	see desc	see desc
ColumnLabel	see desc	see desc
ColumnLabels	HliString *	see desc
AssistOn	see desc	see desc
ColumnsNum	HliSignedShort *	
ColumnsWide	HliSignedShort*	HliSignedShor
ColumnWidths	HliSignedShort *	see desc
ContentsNum	HliUnsignedShort *	
CurrentTool		
ConstantWidth	HliBoolean *	HliBoolean
Cursor	UiCursor *	UiCursor
CursorPosition	HliSignedShort *	HliSignedShort
DecimalPlaces^{cust}	HliSignedShort *	
DefaultBoolean^{cust}	HliBoolean *	HliBoolean
DefaultButton	UiObject *	UiObject
DefaultFloatValue^{cust}	HliDouble *	HliDouble
DefaultIntegerValue^{cust}	HliSignedLong *	HliSignedLong
DefaultItems	HliSignedShort *	see desc
DefaultItemsNum	HliSignedShort *	
DefaultName^{cust}	const char **	const char *
DefaultObject	const char **	const char *
DefaultString^{cust}	HliString	HliConstString
DefaultStringNum	HliSignedShort *	
DefaultValue^{cust}	HliShortString	HliConstShortString
Defective	HliBoolean *	HliBoolean
Dependency	see desc	see desc
DependencyExpression	const char **	const char *
DependencyType	UiDependency *	UiDependency
Editor	const char **	const char *

Table A-1 Attribute Names and TypeSets

Attribute Name	Get	Set
EditorAttributes	see desc	see desc
EditorIcon	UiIcon *	UiIcon
Entry	see desc	
Entry*n*Label	HliString	HliConstShortString
Entry*n*Required	HliBoolean *	HliBoolean
FloatValue	HliDouble *	HliDouble
FontCategory	UiFontCategory *	UiFontCategory
Foreground	UiColor *	UiColor
Framed	HliBoolean *	HliBoolean
Geometry	see desc	see desc
Gutter	HliUnsignedShort *	HliUnsignedShort
Height	UiDimension *	
HeightSpecified	UiDimension *	UiDimension
HelpContext	UiHelpContextPtr	UiHelpContext
HideParent	HliBoolean *	
Icon^{cust}	UiIcon *	UiIcon
IconifiedIcon	UiIcon *	UiIcon
IconifiedTitle	HliString	HliConstString
Inline	HliBoolean *	HliBoolean
InputPermitted	HliBoolean *	HliBoolean
IntegerValue	HliSignedLong *	HliSignedLong
Item	see desc	
ItemsNum	HliSignedShort *	
LabelFontCategory	UiFontCategory *	UiFontCategory
NumColumns	HliUnsignedShort *	HliUnsignedShort *
Orientation	UiOrientation *	UiOrientation
OtherData	void **	void *
OkLabel^{cust}	HliString	HliString
Parent	UiObject *	
Pointer		
PercentDrawn	HliSignedShort *	HliSignedShort
PositionInDesktop	see desc	
Prefilled	UiPrefilled *	UiPrefilled
Prompt		HliConstShortString
PromptOn		see desc
RadioBehavior	HliBoolean *	HliBoolean
Remembered	HliShortString	

Table A-1 Attribute Names and TypeSets

Attribute Name	Get	Set
RememberedBoolean	HliBoolean *	
RememberedItems	HliSignedShort *	
RememberedItemsNum	HliSignedShort *	
RememberedObject	UiObject *	
RememberedString	HliString	
RememberedStringNum	HliSignedShort *	
ResetLabel	HliShortString	HliConstShortString
RowColumn	see desc	see desc
RowColors	see desc	see desc
RowLabel	see desc	see desc
RowLabels	see desc	see desc
RowsNum		
Sensitive	HliBoolean*	HliBoolean
SensitiveNow	HliBoolean *	HliBoolean
Source	const char **	const char *
Specifier	const char **	const char *
StartColumn	HliSignedShort*	HliSignedShort
StartRow	HliSignedShort*	HliSignedShort
State	HliBoolean*	HliBoolean
StatusArea	UiObject *	
Sticky	UiStickyBits*	UiStickyBits
TextFontCategory	UiFontCategory *	UiFontCategory
TextString	HliString	HliConstString
TextStringNum	HliSignedShort *	
TextValue^{cust}	HliShortString	HliConstShortString
Title^{cust}	HliShortString	HliConstShortString
UserText		
Visible		HliBoolean
VisibleNow	HliBoolean *	
VisibleWhenManaged	HliBoolean *	HliBoolean
Width	UiDimension *	
WidthSpecified	UiDimension *	UiDimension
WindowTitle	HliString	HliConstString
XPosition	UiDimension*	
XSpecified	UiDimension*	UiDimension
YPosition	UiDimension*	
YSpecified	UiDimension*	UiDimension

UiActivate

Syntax

```
HliStatus UiActivate(UiObject obj)
```

UiAddColumns

Purpose

Add one or more columns to a scroll matrix.

Syntax

```
HliStatus  UiAddColumns (
Uiobject object,
HliSignedShort position,/* zero-based */
HliString *contents,
HLiString *columnLabels,
HLiSignedShort *columnWidths,
UiAlignment  *alignments,
UiColor *colors,
HliSignedShort num_columns )/* zero-based */
```

Description

Adds new columns to the ScrollMatrix. `Position` indicates the column position before which the new column will be added: 0 adds the column to the beginning, a very high number will cause the column to be added to the end of the ScrollMatrix.

If `columns` is NULL, new empty columns will be added. If `alignments` are NULL, `UI_START_ALIGN` will be used. If `colors` is NULL, the Foreground color attribute will be used. Data from the columns argument will be used to populate the new columns in the ScrollMatrix in row-major order (left to right, top to bottom).

Example

The following example adds 3 columns to a scroll matrix that already had two columns.

```
HliShortString contents[2];
HliCStringToString(”Item1 Added”, contents[0]);
HliCStringToString(”Item2 Added”, contents[1]);
HliCStringToString(”Item3 Added”, contents[2]);

UiAddColumns (
SampleScrollMatrix, 2, contents, NULL, NULL, NULL, NULL,2);
```

UiAddEntry

Purpose

Add an entry or entry list to a menu.

Syntax

```
HliStatus UiAddEntry (
UiObjectobject,
HliSignedShortposition,
UiObjectentry)
```

Description

Adds the entry or entry list `entry` to `object` at `position`**.** Positions are zero origin, so adding at position zero means put at the top of the list. Use a very high position number to guarantee adding to the bottom.

Example

This example assumes that you have created the menu and the example entries and/or entry lists before adding the entries to the menu object.

```
UiObjectent1;
UiObjectcmdent = NULL;
UiObjecttogent = NULL;
UiObjectdirent = NULL;

  //Create entries and the menu object
.
.
.
  UiAddEntry(ent1, 0, cmdent); /* Add to beginning */

  UiAddEntry(ent1, -1, togent); /* Add Entry to end */

  UiAddEntry(ent1, 2, dirent); /* Add Entry to middle */
```

UiAddItem

Purpose

Add entry to a scroll list or an option list.

Syntax

```
HliStatus UiAddItem (
UiObject obj,
HliSignedShort idx, /* zero-based */
HliConstShortString str)/* item to add */
```

Description

Adds the item `str` at the position `idx`. Positions are zero-origin, so adding at position `idx=0` means put at the top of the list. Use -1 to guarantee adding to the bottom of the list.

Example

The following example adds an item whose contents read “Item Added.” The item is added to the end of the scroll list.

```
HliSignedShort pos = -1;
HliShortString hli_item;
HliStatus status;

 status = HliCStringToString(”Item Added”, hli_item);

 status = UiAddItem(ScrollList1, pos, hli_item);
```

Example 2

The example below is from a `.act` file created by the CVact IDT. We can’t encourage you to make system calls because they’re not portable across all CVact platforms, but nevertheless, this example might be informative. It:

- Displays an hour-glass icon to indicate activity
- Sets the scroll list to be invisible
- Reads a directory listing and puts each line into the scroll list

```
FILE *lsin;
HliShortString hlistring;
char line[160];
char *ix;
```

```
UiStartBusy(AlignCmdBox1); /* display hour-glass icon */
UiSetVisibleWhenManaged(ScrollList1,FALSE); /* invisible */
system("ls -a >/tmp/directoryListing");
lsin=fopen("/tmp/directoryListing","r");
ix=fgets(line,80,lsin);
ix=fgets(line,80,lsin);
while(ix){
    line[strlen(line)-1]=NULL;
    HliCStringToString(line,hlistring);
    UiAddItem(ScrollList1,32767,hlistring);
    ix=fgets(line,80,lsin);
    }
fclose(lsin);
UiSetVisibleWhenManaged(ScrollList1,TRUE);
UiEndBusy(AlignCmdBox1);
```

Notes

The scroll list can be either a

- "regular" scroll list (class name is ScrollList), from which the user may select more than one entry
- "radio button" scroll list (class name is RadioScrollList), from which the user may select only one entry

See Also

UiAddItems, UIDeleteItem, UiScrollListCreate, UiRadioScrollListCreate, UiSetChosen, UiGetItem, and UiQueryItem.

UiAddItems

Purpose

Add multiple entries to a scroll list or option list.

Syntax

```
HliStatus UiAddItems (
UiObject obj,
HliSignedShort idx, /* zero-based */
HliString *itms, /* items to add */
HliSignedShort num)/* how many items */
```

Description

Adds `num` number of entries `itms` at the position `idx`. All entries below position `idx` are shifted down to make room.

Positions are zero-based, so adding at position `idx=0` means put at the top of the list. Use -1 to guarantee adding to the bottom of the list.

Example

The example below adds 2 entries.

```
HliSignedShort linesToAdd = 2;/* NOT zero-based */
HliSignedShort posToAddAt = 0;/* zero-based */
HliString itemsText [3];
HliCStringToString(”Line 1 Added”, itemsText[0]);
HliCStringToString(”Line 2 Added”, itemsText[1]);
HliCStringToString(”Line 3 Not Added”, itemsText[2]);
```

```
UiAddItems (
ScrollList1,/* object to add them to */
posToAddAt, /* zero-based */
itemsText, /* text for lines to add */
linesToAdd);/* NOT zero-based (how many items) */
```

Notes

The scroll list can be either a

- “regular” scroll list (class name is ScrollList), from which the user may select more than one entry
- “radio button” scroll list (class name is RadioScrollList), from which the user may select only one entry

See Also

UiAddItems, UIDeleteItem, UiScrollListCreate, UiRadioScrollListCreate, UiSetChosen, UiGetItem, and UiQueryItem.

UiAddRows

Purpose

Add new rows to a scroll matrix.

Syntax

```
HliStatus UiAddRows (
UiObject obj,
HliSignedShort position,
HliString *rowContents,
HliString *rowLabels,
UiColor *colors,
HliSignedShort numRows)
```

Description

`Position` indicates the row position before which the new row will be added. 0 adds the row to the beginning, a very high number will cause the row to be added to the end of the ScrollMatrix.

If `rowsContents` is NULL, new empty rows will be added. If `colors` is null, the Foreground color attribute will be used.

Example

The following example adds three rows to a matrix that has already been defined, created, and initialized.

```
HliStatus status;
HliSignedShort pos = 1;
HliSignedShort numRows = 3;

status= UiAddRows (
SampleScrollMatrix, /* object to add rows to */
pos,/* where to start adding */
NULL,/* no contents for now */
NULL,/* no rowLabels for now */
NULL,/* use default colors */
numRows);/* add this many rows */
```

The following example adds two rows to a matrix that has already been defined, created, and initialized. It also labels the rows.

```
/* add two rows starting at row 2 in the scroll matrix */

HliStatus status;
#define SM2ADDROWS 1 /* zero-based: where to add */
#define NUMROWS 2  /* NOT zero-based: how many to add */
#define SM2COLS 5  /* NOT zero-based */
HliString SM2addRowLabels[SM2ADDROWS];
UiColor SM2addRowColors[SM2ADDROWS *SM2COLS];

/* initialize the row label data for the rows to add */
SM2addRowLabels[0] = HliFromCString("NEW-1");
SM2addRowLabels[1] = HliFromCString("NEW-2");

/* pretend we've initialized the row colors */

status = UiAddRows(SampleScrollMatrix, SM2ADDROWS, NULL,
SM2addRowLabels,SM2addRowColors, NUMROWS);
```

Notes

Data from the `rowContents` argument will be used to populate the new rows in row-major order (left to right, top to bottom).

UiAddSeparator

Purpose

Add a separator after an entry in a menu.

Syntax

```
HliStatus UiAddSeparator(UiObject obj, HliSignedShort pos)
```

Description

Add separator after entry at `pos` in menu `obj`. Positions are zero origin, so adding at position zero means to put at the top of the list.

Example

This example adds a separator after the third entry in the menu named ent1.

```
UiObject ent1;

UiAddSeparator(ent1, 2);
```

UiAligningFormCreate

Purpose

Create an aligning form.

Syntax

```
HliStatus UiAligningFormCreate (
UiObject *obj
HLI_CONST char *name,
UiObject parent,
UiHelpContext help,
HliBoolean framed
HliUnsignedShort ncol,
HliConstShortString title)
```

Description

Create an aligning form of `ncol` columns, with `title` in its title area.

Example

```
Uiobject AlignForm3;
Uiobject DirList;
HliStatus status;
. . .
status = UiAligningFormCreate( &AlignForm3, "AlignForm3"
 ,DirList

/* HelpContext */ ,UI_NULL_HELP
/* Framed */ ,TRUE
/* NumColumns */ ,1
/* Title */ ,HLI_NULL_STRING
);
```

Notes

Do not create an aligning form inside a field.

UiAppDtCreate

Purpose

Create an Application Desktop, which contains Dataset Windows.

Syntax

```
HliStatus UiAppDtCreate
(UiObject       *obj,
HLI_CONST char    *name,
UiHelpContext     help,
HliConstShortString  windowTitle,
UiIcon         iconIcon,
HliConstShortString  iconTitle,
UiCallbackProc     activateCb,
void         *activateData,
UiCallbackProc     helpCb,
void         *helpData,
const char      *currentTool);
```

Description

The input parameters `windowTitle`, `iconIcon`, and `iconTitle` specify window manager attributes. The `activateCb` is called when any of its Dataset Window children receive input focus. `InitialTool` specifies the initial tool to be in effect when the Application Desktop is first created.

Example

UiAppDtCreate

```
(&appDt, ”Application_Desktop”, NULL,
defTitle, defIcon, defIconTitle, activateCb, NULL,
UI_NULL_CALLBACK, NULL, NULL));
```

UiApplicationStateCreate

Purpose

Create an Application State object.

Syntax

```
HliStatus UiApplicationStateCreate
  ((UiObject * obj,
   char *name,
   HliBooleanstate));
```

Description

Create an Application State object named `name` with an initial state of `state`.

Notes

For use in Dependency Expressions.

UiApplyPulldownCreate

Purpose

Create an applied pulldown menu.

Syntax

```
UiApplyPulldownCreate
(UiObject &obj,
char  &name,
UiObject parent,
UiHelpContext help,
HliShortString winTitle,
UiIcon iconIcon,
HliShortString iconTitle,
HliBoolean hideParent)
```

Description

Creates an apply pulldown which is assigned to `obj`. The input parameters `windowTitle`, `iconIcon`, and `iconTitle` specify window manager attributes. The input parameter `parent` specifies the CAB whose apply button will launch this Apply Pulldown. The input parameter `hideParent` specifies whether or not the parent CAB should remain visible when the Apply Pulldown is launched.

Example

```
HliStatus status;
UiObject ApplyPulldn1, CommandBox1;
. . .
status = UiApplyPulldownCreate( &ApplyPulldn1
, "ApplyPulldn1", CommandBox1
/* HelpContext */ ,UI_NULL_HELP
/* WindowTitle */ ,HLI_NULL_STRING
/* IconifiedIcon */ ,UI_NULL_ICON
/* IconifiedTitle */ ,HLI_NULL_STRING
/* HideParent */ ,TRUE
);
```

Notes

The UiObject parent must be a CAB.

UiAssistOn

Purpose

Turn the assist line on or off for all Dataset Windows.

Syntax

```
HliStatus
 ((UiObject obj,/* object for assist
  HliBoolean assistOn));/* turn assist on/off
```

Description

The UiObject obj is an Application Desktop whose assist line is turned on or off according to `assistOn`.

Example

```
UiObject    appDt;
UiAssistOn (appDt, TRUE);
```

Notes

Previously-written help text for the assist line appears when the cursor moves over this object.

See Also

UiPromptOn

UiChooseItem

Purpose

Set an item in an option list to chosen.

Syntax

```
HliStatus UiChooseItem (
UiObject obj,
HliSignedShort position)
```

Description

Set the item obj at `position` to chosen.

Example

This example creates an alignment form and an option list within it, leaving the filling of the option list as an exercise (but see `UiAddItems` and `UiAddItem`, above). It sets the first item in the list so that it defaults to "chosen."

```
UiObject opl1;
UiObjectfaf;
. . .
UiAligningFormCreate
(&faf, /* Returned object handle pointer */
"faf",/* Name of object */
dlb,/* Parent */
UI_NULL_HELP, /* Help context */
FALSE,/* Framed */
2,   /* Number columns */
HLI_NULL_STRING);
UiOptionListCreate
(&opl1, "OpList", faf, UI_NULL_HELP,
UI_NULL_CALLBACK, NULL, HLI_NULL_STRING);
/* missing code to fill option */
. . .
UiChooseItem (opl1, 1));
```

Notes

Making an item chosen makes another item not chosen, because only one item can be chosen at a time.

UiClose

Purpose

Close and dismiss a dialog box or apply pulldown.

Syntax

```
HliStatus UiClose (
UiObject obj )
```

Description

The dialog box obj is forcibly dismissed and the following actions are taken:

- the ClosedCb is called
- the dialog box is made invisible
- VisibleWhenManaged is set to FALSE

For apply pulldowns, obj is closed and disappears. Its parent CommandArgumentBox is not confirmed and is not dismissed. Presumably, this method is invoked if the user has cancelled the operation or otherwise not successfully completed the command.

Example

The following example closes a dialog box named simpleCab. It assumes that the box has already been created.

```
UiObject simpleCab;
. . .
UiClose (simpleCab);
```

Notes

You might want to invoke this function if the user has cancelled the operation or otherwise has not successfully completed a command in a menu entry.

UiCommandBoxCreate

Purpose

Create a top-level, non-aligning form that automatically contains Apply, Reset, and Cancel push buttons.

Syntax

```
HliStatus UiCommandBoxCreate (
UiObject *obj,
HLI_CONST char *name,
UiHelpContext help,
HliConstShortString winTitle,
UiIcon iconIcon,
HliConstShortString iconTitle,
HliBoolean hasApplyButton,
HliConstShortString okLabel,
HliConstShortString applyLabel,
UiCallbackProc applyCb,
void *applyData,
UiCallbackProc cancelCb,
void *cancelData,
HliBoolean modal)
```

Description

Create a Command Argument Box and assign it to obj. The input parameters `windowTitle`, `iconIcon`, and `iconTitle` specify window manager attributes. The input parameters `okLabel` and `applyLabel` specify the label strings to use on the Ok and Apply buttons respectively. The input parameter has `ApplyButton` only has meaning in Motif Look and Feel: if it is set to TRUE, an Apply button will be displayed. The `applyCb` callback will be called whenever the apply or ok button is pressed. The `cancelCb` callback will be called whenever the cancel button is pressed. Setting `modal` to true makes the Command Argument Box modal, which will keep the user from interacting with the rest of the application until this Command Argument Box is dismissed.

Example

The following example creates a Command Argument Box named CommandBox1 and returns its address in Command1.

```
HliStatus status;
UiObject CommandBox1;
. . .
status = UiCommandBoxCreate( &Command1, "CommandBox1"
/* HelpContext */ ,UI_NULL_HELP
```

```
/* WindowTitle */ ,HLI_NULL_STRING
/* IconifiedIcon */ ,UI_NULL_ICON
/* IconifiedTitle */ ,HLI_NULL_STRING
/* ApplyButton */ ,TRUE
/* OkLabel */ ,HLI_NULL_STRING
/* ApplyLabel */ ,HLI_NULL_STRING
/* ApplyCb */ ,UI_NULL_CALLBACK
 ,NULL
/* CancelCb */ ,UI_NULL_CALLBACK
 ,NULL
/* Modal */ ,FALSE
);
```

Notes

There is no parent argument.

UiCommandEntryCreate

Purpose

Create a command entry to be accessed from a Menu or Palette.

Syntax

```
UiCommandEntryCreate
(UiObject *object,
const char *name,
UiHelpContext help,
HliConstShortString assist,

UiKey accelerator,
HliConstShortString textValue,
UiIcon icon,
UiIcon alternateIcon,
UiCallbackProc activateCb,
void *activateCd,
const char *argBox)
```

Description

Create a CommandEntry named `name` and assign it to `object`. The input parameter `accelerator` is a string of the form Ctrl+E for activating the entry via keyboard input. The input parameters `textValue`, `icon` and `alternateIcon` will be used as the label string icon and alternate icon respectively for the entry when it is visualized a in Menu or Palette. An ellipsis (...) will automatically be appended to `textValue` indicating that a Command Argument Box will be launched when this entry is selected. The callback `activateCb` will be called to create the Command Argument Box named `argBox` ONLY if the `argBox` has not yet been created. If the Command Argument Box `argBox` exists it will be launched automatically when this entry is selected, and `activateCb` will not be called.

Example

The following example creates a command entry named item3 and returns a pointer to it in item3ptr. The assist line for this object is initialized to the contents of UIT_APPD_ITEM3_ASSIST, and the text value of the object has been initialized to the contents of UIT_APPD_ITEM3_LABEL.

```
UiObject item3ptr = NULL;
HliShortString item3ast;
HliShortString item3str;
UiCallbackProc menuItemCb;
```

```
/* initialize messages from a header file someplace */
HliGetMessage(UIT_APPD_ITEM3_ASSIST, item3ast);
HliGetMessage(UIT_APPD_ITEM3_LABEL, item3str);
```

UiCommandEntryCreate

```
&item3ptr/* Object id return */
"item3",/* Name of object */
UI_NULL_HELP,/* Help context */
item3ast,/* Assist line */
UI_NULL_KEY,/* Accelerator */
item3str,/* Text value */
UI_NULL_ICON,/* Icon */
UI_NULL_ICON,/* Alternate Icon */
menuItemCb,/* Activate Callback */
NULL,
"simpleCab1");/* Command Argument Box */
```

UiCompletion

Purpose

Complete a Command Argument Box.

Syntax

```
HliStatus UiCompletion(UiObject obj, HliBoolean success)
```

Description

This is usually called from a callback on an apply pulldown push button, to perform the completion normally done when the Apply button in a CAB without an Apply pulldown is pushed. The UiObject obj is a CAB. If `success` is true, the CAB will be confirmed, and, if not defective, remembered values will be saved and the CAB will be closed

Example

This example creates a CAB named simpleCab1 that contains a non-aligning form named `naf`**.** When the user presses the apply button on the CAB, or otherwise completes any of the dialog boxes (if any) contained in the non-aligning form, the `UiCompletion` function marks the command box as complete.

```
HliBoolean bool_complete= TRUE;
UiObject simpleCab1;
UiObject naf;
UiCommandBoxCreate,
(&simpleCab1, ”simpleCab1”, NULL,
cabTitle1,      /* window title */
UI_NULL_ICON,     /* iconified icon */
HLI_NULL_STRING,   /* icon title */
TRUE,         /* hasApplyButton */
okLab,        /* OK label */
applyLab,       /* Apply label */
applyCb, NULL, cancelCb, NULL, FALSE );

UiNonAligningFormCreate
(&naf, ”naf”, simpleCab1, NULL, 10000, 10000);
UiCompletion(simpleCab1, bool_complete);
```

UiConfirm

Purpose

Tell a dialog box to take its confirmation action.

Syntax

```
HliBoolean UiConfirm (
UiObject obj)
```

Description

The dialog box obj is to undertake its “confirmation” action, which typically means to tell its component fields that they are confirmed. For most fields, if they have the UI_REMEMBERED prefill attribute, they will save their current value into their remembered value, so that the next time they are initialized the remembered value will be correct. The dialog box isn’t closed.

For ApplyPulldowns, the CAB parent of the obj pulldown is to be considered confirmed. All its fields are confirmed, which means their remembered values are saved. If the CAB isn’t tacked, it will be dismissed.

Example

The following example confirms the apply pulldown.

```
UiObject appld1;
. . .
UiConfirm(appld1);
```

Ui...CreateFromVStatus

Purpose

Convenience functions that create objects from status code that has been returned by various functions.

Syntax

```
Ui...CreateFromStatus
(UiObject *obj,
.
.
.
/* Ui...CreateFromStatus */
HliStatus status,
. . .)

/* Ui...CreateFromVStatus */
HliStatus status,
va_list args)
```

Description

Looks up the `status code` in the appropriate repository to find the text value, which is returned in `name`. If `args` are provided, the text returned from the repository is treated as a format string and values are substituted. Then the function creates the specified object, obj.

Notes

If you know the `Ui...Create` function for an object, you can obtain the `Ui...CreateFromStatus` signature by substituting the `status` and/or `args` parameters for the last parameter in the `Ui...Create` signature.

UiDatasetWindowCreate

Purpose

Create a dataset window. This is a window which holds presentation areas where the user can work, for example, an area containing a drawing in progress.

Syntax

```
HliStatus UiDatasetWindowCreate
  (UiObject      *obj,
   const char     *name,
   UiHelpContext    help,
   HliConstShortString winTitle,
   UiIcon        iconIcon,
   HliConstShortString iconTitle)
```

Description

In the Single Document Interface model, each Dataset Window will contain a Menu Bar, Presentation Area, and Message Area. The message area displays Assist, Prompt and Status Information. The input parameters `windowTitle`, `iconIcon`, and `iconTitle` specify window manager attributes.

Example

The following example creates a dataset window that uses a martini as an icon, and preset titles for the window and the iconified icon.

```
UiObject datasetWindow;
HliShortString defTitle;
HliShortString defIconTitle;
UiIcon martini;

  HliGetMessage(UIT_APPD_DEFTITLE,defTitle);
  HliGetMessage(UIT_APPD_DEFICONTITLE, defIconTitle);
```

UiDatasetWindowCreate `(&datasetWindow, "DSW_1", NULL, defTitle, martini, defIconTitle);`

UiDeleteColumns

Purpose

Delete columns from a scroll matrix.

Syntax

```
HliStatus UiDeleteColumns (
UiObject obj,
HliSignedShort position,
HliSignedShort columns_num)
```

Description

Deletes the number of columns specified by `columns_num` in the position array starting at `position`.

UiDeleteEntry

Purpose

Delete an entry from a menu.

Syntax

```
HliStatus UiDeleteEntry(UiObject obj, UiObject entry)
```

Description

Deletes the entry or entry list `entry` from obj**.** All other entries move up one position.

Example

The following example assumes that the CommandEntry named `cmdent` has been created. The example shows how to obtain the pointer to the entry that contains the text “entryList1” and how to delete the entry at that position.

```
UiObject ent1;
UiObject cmdent = NULL;
. . .
UiObtainEntry(”entryList1”,&ent1);
UiDeleteEntry(ent1, cmdent);
```

See Also

UiAddEntry

UiDeleteItem

Purpose

Delete an item from a scroll list or option list.

Syntax

```
HliStatus UiDeleteItem (
UiObject obj,
HliSignedShort idx)
```

Description

Deletes the item specified by `idx` from the list. All other items move up one position.

Example

The following example deletes the 10th item in the scroll list named ScrollList1.

```
Uiobject ScrollList1;
. . .
UiDeleteItem(ScrollList1,9);
```

See Also

UiAddItem

UiDeleteRows

Purpose

Delete row(s) from a scroll matrix.

Syntax

```
HliStatus UiDeleteRows (
UiObject obj,
HliSignedShort position,
HliSignedShort numRows)
```

Description

Deletes the number of rows specified by `numRows` in the matrix starting at `position` (0-based).

UiDeleteSeparator

Purpose

Delete a separator in a menu.

Syntax

```
HliStatus UiDeleteSeparator(
UiObject obj,
HliSignedShort pos)
```

Description

Delete the separator at `pos` from the menu obj.

Example

The following example assumes that the menu object named ent1 has already been created and initialized. It deletes the 3rd entry in it.

```
UiObject ent1;
. . .
UiAddSeparator(ent1, 2);
. . .
UiDeleteSeparator(ent1, 2);
```

See Also

UiAddSeparator

UiDestroy

Purpose

Destroy a container and its children.

Syntax

```
HliStatus UiDestroy(UiObject obj)
```

Description

This function destroys the container's children first. Then it calls the container's DestroyCb callback, if any. After that, it destroys itself.

UiDialogBoxCreate

Purpose

Create a dialog box.

Syntax

```
HliStatus UiDialogBoxCreate (
UiObject *obj,
const char *name,
UiHelpContext help,
HliConstShortString winTitle,
UiIcon iconifiedIcon,
HliConstShortString iconTitle,
HliBoolean modal)
```

Description

Create a dialog box and assigns it to obj**.**

Example

```
Uiobject DirList;
HliStatus status;

. . .
status = UiDialogBoxCreate( &DirList, "DirList"
/* HelpContext */ ,UI_NULL_HELP
/* WindowTitle */ ,HLI_NULL_STRING
/* IconifiedIcon */ ,UI_NULL_ICON
/* IconifiedTitle */ ,HLI_NULL_STRING
/* Modal */  ,FALSE
);
```

UiDirectEntryCreate

Purpose

Invoke an immediate action.

Syntax

```
HliStatus UiDirectEntryCreate
(UiObject * object,
 const char *name,
UiHelpContext help,
HliConstShortString assist,
UiKey accelerator,
HliConstShortString textValue,
UiIcon icon,
UiIcon alternateIcon,
UiCallbackProc activateCb,
void *activateCd)
```

Description

Create a direct action entry and assign it to `object`**.** Selection of this entry causes the ActivateCb to be invoked.

Notes

The ActivateCb is inherited from the Entry Class.

UiEditCell

Purpose

Display a cell from a scroll matrix so that the user can edit it.

Syntax

```
HliStatus UiEditCell (
UiObject *obj,
HliSignedShort row
HliSignedShort column
HliString *value)
```

Description

Allows the application developer to programmatically force a specific cell to be edited. This function will first attempt to commit the edit in the current cell. If the commit is disallowed then `UiEditCell` will return, otherwise the specified cell at `row`, `column` is scrolled until it is visible, then its starting `value` is displayed.

UiEditorDialogBoxCreate

Purpose

Create an editor dialog box and assign it to `ed_uiobj`.

Syntax

```
UiEditorDialogBoxCreate
(UiObject *ed_uiobj,
const char *name,
HliConstShortString winTitle,
UiHelpContext help,
UiIcon iconIcon,
HliConstShortString iconTitle,
HliBoolean modal)
```

UiEndBusy

Purpose

Remove an hourglass cursor.

Syntax

```
HliStatus UiEndBusy( )
```

Description

`UiEndBusy` works with `UiStartBusy` to indicate that the application is busy or not.

Example

```
HliStatus status;
. . .
status = UiEndBusy ;
```

See Also

UiStartBusy

UiEntryListCreate

Purpose

Create an EntryList.

Syntax

```
UiEntryListCreate (
UiObject *obj,
const char *name,
UiHelpContext help,
HliConstShortString assist,
HliConstShortString text,
HliBoolean isInline)
```

UiErrorBoxCreate
UiErrorBoxCreateFromStatus
UiErrorBoxCreateFromVStatus

Purpose

Create an error box.

Syntax

```
UiErrorBoxCreate
(UiObject *obj,
const char *name,
UiHelpContext help,
UiIcon iconIcon,
HliConstShortString iconTitle,
UiCallbackProc okCb,
void *okData,
HliString text_string)
```

Description

Creates an error box and assigns it to obj.

UiFailureBoxCreate

Purpose

Create a failure box to warn the user that an operation has failed.

Syntax

```
UiFailureBoxCreate
(UiObject *obj,
const char *name,
UiHelpContext help,
UiIcon iconIcon,
HliConstShortString iconTitle,
UiCallbackProc okCb,
void *okData,

UiCallbackProc reportCb,
void *reportData,
HliString admintext_string,
HliString text_string)
```

Description

Creates a failure box and assigns it to obj.

UiFillinCreate

Purpose

Create a fillin field.

Syntax

```
HliStatus UiFillinCreate (
UiObject *obj,
const char *name,
UiObject parent,
UiHelpContext help,
char *editor,
UiCallbackProc completeCb,
void *completeCd,
HliConstString dvalue)
```

Description

Creates a text fillin and assigns it to obj.

UiFillinStrlen

Purpose

Returns the true visible string length for Fillin, IntFillin and FloatFillin fields. Finds true visible text length of Fillin, IntegerFillin or FloatFillin fields.

Syntax

```
HLI_EXT HliSignedLong UiFillinStrlen HLI_ARGS ( (UiObject obj) );
```

Description

The UiTextString binding does not return the correct visible string length in the case of an IntFillin or a FloatFillin when they are empty. When IntFillin or FloatFillin is empty, the string “0”or “0.0000” is returned. Hence you cannot determine whether an IntFillin or FloatFillin is really empty or not.

For IntFillin or FloatFillin, UiGetTextString returns “0” or “0.0000” even when the IntFillin or FloatFillin is physically empty. This is because an IntFillin or FloatFillin is meant for numeric input and is never supposed to be empty.

There are cases where it is essential to know whether an IntFillin or FloatFillin is empty or not. The UiFillinStrlen always returns the length of the visible string. It returns 0 if IntFillin or FloatFillin is empty.

Notes

Returns the following:

- Visible string length if successful
- -1 if not successful

UiFloatFillinCreate

Purpose

Create a fillin field to contain floating point values.

Syntax

```
HliStatus UiFloatFillinCreate (
UiObject *obj,
HLI_CONST char *name,
UiObject parent,
UiHelpContext help,
UiCallbackProc completeCb,
void *completeCd,
HliDouble defaultValue,
HliDouble minVal,
HliDouble maxVal,
HliSignedShort decimals)
```

Description

Creates a float fillin field and assigns it to obj.

UiGet...

Purpose

Get attribute for a “simple” object.

Syntax

```
HliStatus UiGetAttributeName(
UiObject obj,
AttributeTypeDef AttributeValue)
```

Description

Most Ui object attribute values can be retrieved according to the above syntax, if you know the attribute’s name, type, and whether to use a pointer to receive its value or a pointer to the pointer to the value. We list this information in the table at the beginning of this chapter.

Exceptions to this include objects such as the following, which often take more than two arguments in their get and set functions:

- callbacks (see `UiGet...Cb`)
- option lists (see `UiGetAttributeName`)
- scroll matrixes (see `UiGetAttributeName`)
- apply pulldowns (see `UiGetEntryRequired`)

For the multi-parameter attributes within those objects, we provide separate entries.

See Also

Attribute Table

UiGet...Cb

Purpose

Get pointer to a callback procedure for an object.

Syntax

```
HliStatus UiGetCallBackNameCb(
UiObject obj,
UiCallbackProc *callback,
void **callbackdata)
```

Description

The callback procedures can all be obtained by using the `UIGet...Cb` function shown above.

Table A-2 Callback Names

Name	Name
AbortCb	EditorFocusChangeCb
ActivateCb	EditorInvokedCb
ApplyCb	EditorRescindCb
CancelCb	Entry*n*Cb [see Note]
ChosenCb	EntryCb
ClosedCb	InitializeCb
CompleteCb	InputCb
CompleteCellCb	IsDropOkCb
ConfirmCb	IsEditableCb
DeactivateCb	IsInputOkCb
DefaultActionCb	MouseEventCb
DestroyCb	PropertiesCb
DismissCb	PushedCb
DropCb	ResetCb

Notes

For apply pulldowns, the following call is also available to get the callback procedure and data for entry **i** in Apply Pulldown obj.

```
HliStatus UiGetEntryCb
 ((UiObject obj,
  HliSignedShort i,
  UiCallbackProc *proc,
  void **data));
```

UiGetCell

Purpose

Get the value of a cell in a scroll matrix.

Syntax

```
HliStatus UiGetCell (
UiObject obj,
HliSignedShort row,
HliSignedShort column,
HliString value)
```

Description

Copies the value of the string in the cell at `row`, `column` into `value`.

UiGetCellColor

Purpose

Get the color value of a cell in a scroll matrix.

Syntax

```
HliStatus UiGetCellColor (
UiObject obj,
HliSignedShort row,
HliSignedShort column,
UiColor *color)
```

Description

Returns in `color` the color value of the specified matrix cell at `row`, `column`.

Notes

To get many cells, it may be more efficient to get the Color attribute.

UiGetColumnColors

Purpose

Get the color values of a column in a scroll matrix.

Syntax

```
HliStatus UiGetColumnColors (
UiObject obj,
HliSignedShort idx,
UiColor *colors)
```

Description

Returns the color values of an entire column. `Idx` indicates the position of the column whose colors are to be retrieved.

UiGetColumnLabel

Purpose

Get the label for the specified column in a scroll matrix.

Syntax

```
HliStatus UiGetColumnLabel (
UiObject obj,
HliShortString columnLabel,
HliSignedShort idx)
```

Description

Returns in `columnLabel the` label of the column specified by `idx` in the scroll matrix.

See Also

UiGet...

UiGetDependency

Purpose

Syntax

```
HliStatus UiGetDependency (
UiObject obj,
UiDependency *type,
const char **expression)
```

See Also

UIGet...

UiGetEntry

Purpose

Retrieve an entry or entry list from a specific position in a menu.

Syntax

```
HliStatus UiGetEntry (
UiObject obj,
HliSignedShort pos,
UiObject *entry)
```

Description

Retrieve the `entry` (or entry list) in the `pos` position in the menu obj.

UiGetEntryLabel

Purpose

Retrieve the label of an entry at a given position in an apply pulldown.

Syntax

```
HliStatus UiGetEntryLabel
 ((UiObject obj,
  HliSignedShort i,
  HliString buf))
```

Description

Return in `buf` the label string for entry `i` in Apply Pulldown obj.

UiGetEntryRequired

Purpose

Retrieve the required flag for an entry in a pulldown.

Syntax

```
HliStatus UiGetEntryRequired
 ((UiObject obj,
  HliSignedShort i,
  HliBoolean *pRequired));
```

Description

Return the required flag for the specified entry in Apply Pulldown obj.

UiGetFocus

Purpose

Get the focus state of the fillin field or paragraph.

Syntax

```
HliBoolean UiGetFocus (
UiObject obj // Fillin field or paragraph object ONLY.
)
```

Description

Use this function to get the current focus state of the fillin field or paragraph only.

- When a fillin field or paragraph gets focus, TRUE is returned.
- When a fillin field or paragraph loses focus, FALSE is returned

Example

```
UiObject fillin;
HliBoolean focusState;

focusState = UiGetFocus ( fillin );
```

Notes

Valid for integer fillin, float fillin, text fillin or paragraph.

UiGetGeometry

Purpose

Get the size and coordinates of an object.

Syntax

```
HliStatus UiGetGeometry (
UiObject obj,
UiDimension *x,
UiDimension *y,
UiDimension *width,
UiDimension *height)
```

Description

Return the current `x` and `y` position, relative to its parent, and the current width and height for the UiObject obj.

UiGetGeometrySpecified

Purpose

Get the size and position of an object.

Syntax

```
HliStatus UiGetGeometrySpecified (
UiObject obj,
UiDimensionPtr x,
UiDimensionPtr y,
UiDimensionPtr width,
UiDimensionPtr height )
```

Description

Return the user specified `x` and `y` position, relative to its parent, and the user specified width and height for the UiObject obj.

UiGetInstanceId

Purpose

Gets the instance id of the specified UiObject which is the instance of a specified dialog or property sheet. You can use `UiGetInstanceId` to get the instance id for each of the multiple instances of a CVact menu.

Syntax

```
HliStatus UiGetInstanceId
(UiObject obj,
HliUnSignedShort *inst_id);
```

Description

The Cl function `ClRegisterDialogMaxInstances` allows the creation of multiple instances of the specified property sheet or dialog box. The maximum number of instances allowed for a property sheet or dialog box is in the range 1 to 65,535. If the maximum number of instances already exist, a new instance is not created.

The instance id is in the range 1 to 65,535. The Cl Library automatically sets the appropriate instance id on each of the multiple instances of the dialog box or property sheet which are launched from the CADDS algorithms using `cl_launch_dialog` or from the CR side using the `ClLaunchDialog` binding. In a session, the instance id, once assigned, remains the same and does not change.

Please note: Do not use `UiSetInstanceId` to assign an id to an instance of a property sheet or dialog box if you have used `cl_launch_dialog` or `ClLaunchDialog` to launch the dialog box or property sheet already registered with `ClRegisterDialog`.

Example

```
HliUnsignedShort inst_id
UiGetInstanceId (CamuViewUnviewComponentCab;
& inst_id);
```

See Also

UiSetInstanceId

UiGetItem

Purpose

Get an option list's item's text value and chosen state.

Syntax

```
HliStatus UiGetItem (
UiObject obj,
HliSignedShort idx,
HliShortString str,
HliBoolean *chosen)
```

Description

Returns in chosen whether or not the item at position `idx` has been chosen. Returns its text in `str`.

See Also

UiGet...

UiGetItemSensitive

Purpose

Get the sensitive state of the specified option list item.

Syntax

```
HliStatus UiGetItemSensitive
(UiObject obj, HliSignedShort item,
HliBoolean *state
);
```

Description

Queries the sensitive state of the option list item.

TRUE indicates that the item can obtain focus, and permits user input (if appropriate for this object). FALSE indicates that the item is hazed out, prevents user input, and disallows focus.

UiGetObjectByName

Purpose

Gets the UiObject given its name.

Syntax

```
HliStatus UiGetObjectByName
(const char *name, UiObject parent, UiObject *obj);
```

Description

Converts a string representing the name of an object to an object. If the syntax is incorrect, an error status is returned. If the syntax is correct, but no such object currently exists, a warning status is returned.

If the parent field is not null, the function checks the children of the parent for an object with the specified name and returns a reference to it if it is found, or null if not found. If the parent field is null, it assumes that the name is fully qualified and effectively searches the entire instance tree.

`UiGetObjectByName` also works with multiple instances of a dialog box or property sheet. For dialogs having multiple instances, `UiGetObjectByName` works only on top level dialogs.

For a dialog box or property sheet having multiple instances:

- Call the Cl function `ClGetDialogNumCurInstances` to get the current number of instances of the property sheet or dialog box.
- Allocate a UiObject array of size `(number of current instances * sizeof (UiObject)`.
- Invoke `UiGetObjectByName` to locate all instances of the property sheet or dialog box.
- Pass the UiObject array pointer as the obj parameter to the `UiGetObjectByName` call.

  `UiGetObjectByName` treats the obj parameter as an array of UiObjects for dialog boxes or property sheets having multiple instances. An array of all instances of the specified dialog box or property sheet is returned using the obj parameter.

To locate an object inside a particular instance of a property sheet or dialog box, first locate the appropriate top level instance using `UiGetObjectByName` as specified earlier.Then, again using `UiGetObjectByName,` with the parent specified as the right instance, search for the secondary level object.

UiGetPositionInDesktop

Purpose

Get an object's position in the desktop.

Syntax

```
HliStatus UiGetPositionInDesktop (
UiObject obj,
UiDimension *x,
UiDimension *y)
```

Description

Return the `x` and `y` position, relative to its parent, for the UiObject obj.

UiGetResetInProgress

Purpose

Determines whether a reset is in progress for a Command Argument Box(CAB).

Syntax

```
HliStatus UiGetResetInProgress (
UiObject obj,
HliBoolean *rflag);
```

Description

A Command Argument Box (CAB) requires the reset in progress status so that the InitializeCb callback can take different actions when the actual reset is in progress. InitializeCb gets called for the following two reasons:

- When the Command Argument Box (CAB) becomes visible
- When the Reset button on the CAB is pressed

InitializeCb is called each time the ResetCb is issued. ResetCb is called when the Reset button on the CAB is pressed to reset the contents of the CAB.

The following two callbacks may also require the reset in progress status:

- CompleteCb
- ChosenCb

Please note: The object should be of type Command Argument Box.

The `rflag` argument is filled up with the reset in progress status which is a HliBoolean value.

UiGetRowColors

Purpose

Get the color values of an entire row in a scroll matrix.

Syntax

```
HliStatus UiGetRowColors (
UiObject obj,
HliSignedShort idx,
UiColor *colors)
```

Description

Retrieves the color values of an entire row. `Idx` indicates the position of the row whose colors are to be retrieved.

UiGetRowColumn

Syntax

```
HLI_EXT HliStatus UiGetRowColumn HLI_ARGS
    ((UiObject obj,
      HliSignedShort* col,
      HliSignedShort* wide,
      HliSignedShort* row,
      HliSignedShort* high));
```

UiGetRowLabel

Purpose

Get the label for the specified row in a scroll matrix

Syntax

```
HliStatus UiGetRowLabel (
UiObject obj,
HliShortString rowLabel,
HliSignedShort idx)
```

Description

Return the label in `rowLabel` of the `idx`'th row.

UiGetRowLabels

Purpose

Get all the row labels from a scroll matrix.

Syntax

```
HliStatus UiGetRowLabels (
UiObject obj,
HliString *rowLabels)
```

Description

Return an array of labels in `rowLabels`.

UiIdentificationBoxCreate

Purpose

Create an identification box to identify an application and give information such as copyright, version, etc.

Syntax

```
HliStatus UiIdentificationBoxCreate (
UiObject *obj,
const char *name,
UiHelpContext help,
HliString winTitle,
UiIcon iconIcon,
HliConstShortString iconTitle,
HliConstShortString appname,
HliConstShortString vernum,
HliConstShortString copyright,
HliConstShortString vendorname,
const char *iconname,
UiCallbackProc okCb,
void *okData )
```

UiInformationBoxCreate UiInformationBoxCreateFromStatus UiInformationBoxCreateFromVStatus

Purpose

Create an information box to display messages, but not warnings or errors.

Syntax

```
HliStatus UiInformationBoxCreate(
UiObject *obj,
HLI_CONST char *name,
UiHelpContext help,
UiIcon iconIcon,
HliConstShortString iconTitle,
UiCallbackProc okCb,
void *okData,


/* UiInformationBoxCreate */
HliString text_string)

/* UiInformationBoxCreateFromStatus */
HliStatus status,
. . .)

/* UiInformationBoxCreateFromVStatus */
HliStatus status,
va_list args)
```

Description

Create an information box and assigns it to obj.

Notes

If you know the `Ui...Create` function for an object, you can obtain the `Ui...CreateFromStatus` signature by substituting the status and/or `args` parameters for the last parameter in the `Ui...Create` signature.

UiIntegerFillinCreate

Purpose

Create a fillin field for integers.

Syntax

```
HliStatus UiFillinCreate (
UiObject *obj,
const char *name,
UiObject parent,
UiHelpContext help,
char *editor,
UiCallbackProc completeCb,
void *completeCd,
HliConstString dvalue)
```

Description

Creates an integer fillin field and assigns it to obj.

UiIsDefaultButton

Purpose

Checks to see if the given Pushbutton is the default button on a dialog box.

Syntax

```
HliBoolean UiIsDefaultButton (UiObject obj, UiObject db);
```

Description

Checks if the given Pushbutton has been set as the `DefaultButton` attribute of the dialog box.

Refer to “UiSetDefaultButtonWithSThickness” on page A-115 .

UiLabelCreate

Purpose

Create a label for an object.

Syntax

```
HliStatus UiLabelCreate (
UiObject *obj,
const char *name,
UiObject parent,
HliConstShortString label,
UiIcon icon)
```

Description

Creates a label and assigns it to obj**.**

Example

```
HliStatus status;
UiObject Label1, AF1;
. . .
status = UiLabelCreate( &Label1, ”Label1”, AF1
/* TextValue */ ,”My Label’s Displayed Name”
/* Icon */ ,UI_NULL_ICON
);
```

UiNonAligningFormCreate

Purpose

Create a non-aligning form.

Syntax

```
HliStatus UiNonAligningFormCreate (
UiObject *obj,
HLI_CONST char *name,
UiObject parent,
UiHelpContext help,
UiDimension width,
UiDimension height)
```

Description

Creates a non-aligning form and assigns it to obj.

Example

```
HliStatus status;
UiObject NAF1, CommandBox1;
. . .

status = UiNonAligningFormCreate( &NAF1, ”NAF1”, CommandBox1 /*
HelpContext */ ,UI_NULL_HELP
/* WidthSpecified */ ,UI_UNSPECIFIED_DIMENSION
/* HeightSpecified */ ,UI_UNSPECIFIED_DIMENSION
);
```

UiObtainEntry

Purpose

Assign an entry to a menu object.

Syntax

```
HliStatus UiObtainEntry (
const char *name,
UiObject *entry)
```

Description

This function assigns to entry the entry instance which corresponds to the specified name. If the entry doesn't exist, the EntryDefiner is called to create it.

UiOptionListCreate

Purpose

Create an option list.

Syntax

```
HliStatus UiOptionListCreate (
UiObject*obj,

HLI_CONST char*name,
UiObject parent,
UiHelpContext helpContext,
UiCallbackProc procChosenCb,
void *dataChosenCb,
HliConstShortString defaultValue )
```

Description

Creates an option list and assigns it to obj.

UiPaletteCreate

Purpose

Create an icon palette.

Syntax

```
HliStatus UiPaletteCreate
(UiObject obj,
char *name,
UiHelpContext help_context,
HliConstShortString iconified_text,
UiIcon iconified_icon,
HliUnsignedShort num_columns)
```

Description

Creates a palette and assigns it to obj. You can specify the name of the object, the `help_context` to provide help for it, the text and icon to use when the object is iconified, and the number of columns in the palette.

UiParagraphCreate

Purpose

Create a paragraph or multi-line text area.

Syntax

```
HliStatus UiParagraphCreate (
UiObject *obj,
HLI_CONST char *name,
UiObject parent,
UiHelpContext help,
HliSignedShort max_length,
UiCallbackProc completeCb,
void *completeCd,
HliConstString default_text)
```

Description

You can specify the `max_length` in lines and initialize the paragraph with the `default_text`**.** You can register a callback (`completeCb`) to be called, and callback data (`completeCd`) when the user closes the paragraph. You can also specify help text, the name of the paragraph, and the paragraph's parent's name.

UiParagraphInsertText

Purpose

To insert text in a paragraph at the current cursor location.

Syntax

```
HliStatus UiParagraphInsertText (
UiObject obj, // paragraph object ONLY.
char* text // Text to be inserted at current cursor location.
)
```

Description

You can insert specified text at any location of the cursor in the paragraph.

Example

```
UiObject para;
char* str = "text to be inserted"; HliStatus st;
st = UiParagraphInsertText ( para, str );
```

Notes

Valid for paragraphs only.

UiParagraphMoveCursor

Purpose

To move the cursor in a paragraph.

Syntax

```
HliStatus UiParagraphMoveCursor (
UiObject obj, // paragraph object ONLY.
char* direction // Text to be inserted at current cursor location.
)
```

Description

Use this function to move the cursor in a paragraph in a manner specified by the direction string.

The valid values for direction are:

- Up : Move cursor up by one line
- Down : Move cursor down by one line
- Left : Move cursor to the left by one character
- Right : Move cursor to the right by one character
- PgUp : Scroll up
- PgDn : Scroll down
- Backspace : Erase character before the cursor location

Example

```
UiObject para;
char* dir = "Up";
HliStatus st;
st = UiParagraphMoveCursor ( para, dir );
```

Notes

Valid for only a paragraph object. The function argument (direction) is not case sensitive. Use this function for applications like callback for a button indicating direction.

UiPostCab

Purpose

Post a dialog box and wait until it is dismissed.

Syntax

```
HliStatus UiPostCabAndWait(UiObject obj, HliUnsignedShort
*btnflag);
```

Description

Post the cab (Modal Dialog box) pointed to by `obj` and wait for the cab to be dismissed (that is, when `*btnflag` becomes true). When you click the OK or Cancel button, set the flag to `true`. This enables sequential processing of the cabs.

Example

```
UiObject obj2;
static HliUnsignedShort btnflag;
            .
            .
            .
/* obj2 points to the Dialog Box cab to be posted */
            .
            .
            .
UiPostCabAndWait(obj2, &btnflag);
```

UiPromptOn

Syntax

```
HliStatus UiPromptOn (
UiObject obj,
HliBoolean promptOn )
```

UiPushButtonCreate

Purpose

Create a push button.

Syntax

```
HliStatus UiPushButtonCreate (
Uiobject *obj,
HLI_CONST char *name,
UiObject parent,
UiHelpContext help,
HliConstShortString assist,
HliConstShortString label,
UiIcon icon,
UiIcon altIcon,
UiCallbackProc pushedCb,
void *pushedCb )
```

Example

```
Uiobject PushButton3, AlignForm4;
HliStatus status;
. . .
status = UiPushButtonCreate( &PushButton3, ”PushButton3”
, AlignForm4
/* HelpContext */ ,UI_NULL_HELP
/* AssistLine */ ,HLI_NULL_STRING
/* TextValue */ ,”PushMe”
/* Icon */ ,UI_NULL_ICON
/* AlternateIcon */ ,UI_NULL_ICON
/* PushedCb */ ,PushedCb_PushButton3
 ,NULL
);
```

UiQueryItem

Purpose

Find the position and chosen state of an item in a scroll list.

Syntax

```
HliStatus UiQueryItem (
UiObject obj,
HliConstShortString str,
HliSignedShort *pos,
HliBoolean *chosen)
```

Description

Given `str` the text value of an item in a scroll list, return its position in `pos` and whether or not it's chosen in chosen.

Example

```
HliString  Value ;
HliBoolean   Chosen ;
HliSignedShort Pos ;
UiObject    Obj ;
HliStatus   st ;

. . .
st = UiQueryItem( Obj, Value, &Pos, &Chosen ) ;

  if ( st == HLI_STATUS_OK )
    return ( TRUE ) ;  /* Value was found */
  else
    return ( FALSE ) ;  /* Value was not found */
```

Notes

If there are multiple items with the same text value, the first is the one found.

UiQueryItmStatusByIdx

Purpose

Query the chosen status of an item in a scroll list when the scroll list has multiple entries with identical item name.

Syntax

```
HliStatus UiQueryItmStatusByIdx
(UiObject obj,
HliSignedShort idx,
HliBoolean *chosen);
```

Description

Given an item number index in a scroll list assigned `obj`, returns its chosen state in `chosen`.

Example

```
HliBoolean   Chosen ;
HliSignedShort idx;
UiObject    Obj ;
HliStatus   st ;
/* obj is required to be Scroll list */
. . .
st = UiQueryItmStatusByIdx( Obj, idx, &Chosen ) ;

  if ( st == HLI_STATUS_OK ) {
if ( chosen == TRUE ) /* Item is chosen*/
/* Do processing for item chosen*/
  else
/* Do processing for item not chosen*/
}
```

Notes

If there are multiple items with the same name in the scroll list, the correct chosen state of an item number is reported. `UiQueryItem` and `UiGetItem` does not report the correct chosen state of an item if the scroll list has multiple entries.

UiQueryResultBoxCreate

Purpose

Create a query result box.

Syntax

```
HliStatus UiQueryResultBoxCreate
(UiObject *obj,
const char *name,
UiObject parent,
UiHelpContext help,
HliConstShortString assist,
HliConstShortString label,
UiIcon icon,
UiIcon altIcon,
UiCallbackProc pushedCb,
void *pushedCd)
```

Description

Creates a query result box and assigns it to obj.

UiQuestionBoxCreate
UiQuestionBoxCreateFromStatus
UiQuestionBoxCreateFromVStatus

Purpose

Create a question box, used to ask the user a yes or no question.

Syntax

```
HliStatus UiQuestionBoxCreate (
UiObject *obj,
char *name,
UiHelpContext help,
UiIcon iconIcon,

HliConstShortString iconTitle,
UiCallbackProc yesCb,
void *yesData,
UiCallbackProc noCb,
void *noData,

/* UiQuestionBoxCreate */
HliString text_string )

/* UiQuestionBoxCreate */
HliStatus status,
. . .)

/* UiQuestionBoxCreate */
HliStatus status,
va_list args)
```

Description

Creates a question box and assigns it to obj.

Notes

If you know the `Ui...Create` function for an object, you can obtain the `Ui...CreateFromStatus` signature by substituting the status and/or `args` parameters for the last parameter in the `Ui...Create` signature.

UiRadioScrollListCreate

Purpose

Create a scroll list with radio behavior.

Syntax

```
HliStatus UiRadioScrollListCreate (
UiObject *obj,
const char *name,
UiObject parent,
UiHelpContext help,
UiCallbackProc completeCb,
void *completeCd,
HliConstShortString defaultValue)
```

Description

This is a convenience function that creates a scroll list with radio behavior. There is no class RadioScrollList.

Notes

Sets RadioBehavior to TRUE.

UiRadioSetCreate

Purpose

Create an empty set of radio buttons.

Syntax

```
HliStatus UiRadioSetCreate (
UiObject *obj,
HLI_CONST char *name,
UiObject parent,
UiHelpContext help,
HLI_CONST char *defaultName)
```

Example

```
HliStatus status;
UiObject RadioSet1;
. . .
status = UiRadioSetCreate( &RadioSet1, ”RadioSet1”, AF1
/* HelpContext */ ,UI_NULL_HELP
/* DefaultName */ ,NULL
);
```

Notes

Toggles should then be created and parented within this set.

See Also

UiToggleCreate

UiReset

Purpose

Reset a dialog box.

Syntax

```
HliStatus UiReset (
UiObject obj)
```

Description

The dialog box is to undertake its “reset” action, which typically means to tell its component fields that they are reset. For most fields, this means that they will reset their value by copying from their default or remembered value, or they will invoke their initialization callback.

If `UiReset` is called from a callback, then another callback can be invoked while the first one is in progress.

Notes

If the dialog box was tacked before `UiReset` was called, the dialog box continues to remain tacked.

UiResetCab

Purpose

Reset a command box.

Syntax

```
HliStatus UiResetCab (
UiObject obj // Command Box object only
)
```

Description

Use this function to reset the entire command box. This typically means that all its component fields are also reset. That is, all the component fields of the command box reset their value by copying their default value. Or, they invoke their initialization callback.

Please note: If the command box is already tacked when this function is called, the command box continues to remain tacked.

Example

```
UiObject cab;;
HliStatus st;

st = UiResetCab ( cab );
```

Notes

Valid for objects of type Command Box.

UiScrollListCreate

Purpose

Create a scroll list

Syntax

```
HliStatus UiScrollListCreate (
UiObject *obj,
HLI_CONST char *name,
UiObject parent,
UiHelpContext help,
UiCallbackProc completeCb,
void * completeCd,
HliConstShortString initValue)
```

Description

Creates a scroll list and assigns it to obj.

Example

```
Uiobject ScrollList1, AlignForm3;
HliStatus status;
. . .
status = UiScrollListCreate( &ScrollList1, ”ScrollList1”
, AlignForm3
/* HelpContext */ ,UI_NULL_HELP
/* CompleteCb */ ,UI_NULL_CALLBACK
 ,NULL
/* DefaultValue */ ,NULL
);
```

UiScrollMatrixCreate

Purpose

Creates an empty scroll matrix with the specified number of columns and rows.

Syntax

```
HliStatus UiScrollMatrixCreate (
UiObject* obj,
HLI_CONST char *name,
UiObjectparent,
UiHelpContexthelp,
HliSignedShort rows,
HliSignedShort columns,
HliSignedShort *colWids,
UiCallbackProc completeCb,
void *completeCd)
```

Description

Creates a scroll matrix and assigns it to obj.

UiSeparatorCreate

Purpose

Create a separator.

Syntax

```
HliStatus UiSeparatorCreate (
UiObject *obj,
char *name,
UiObject parent,
UiOrientation orient)
```

UiSet...

Purpose

Set attribute for a simple object.

Syntax

```
HliStatus UiSetAttributeName(
UiObject obj,
AttributeTypeDef AttributeValue)
```

Description

Most Ui object attribute values can be set according to the above syntax, if you know the attribute's name and type We list this information in the table at the beginning of this chapter.

Exceptions to this include objects such as the following, which often take more than two arguments in their get and set functions:

- callbacks (see `UiSet...Cb`)
- option lists (see `UiSetAttributeName`)
- scroll matrixes (see `UiSetAttributeName`)

For the multi-parameter attributes within those objects, we provide separate entries.

See Also

Attribute Table

UiSetBorderWidthForSldoffDlgBox

Purpose

Allows setting of border width for a Slideoff Dialog Box.

Syntax

```
HliStatus UiSetBorderWidth (
UiObject obj, // Slideoff Dialog Box Object ONLY
HliUnsignedShort border_width)
```

Description

A border of width 12 units exists by default for Slideoff dialog boxes. This is required for the look and feel of the CADDS calculator. The top bar pulldowns in the Interactive Surface Design environment do not need a border. A border width value of 0 removes the border.

Example

The following example sets the border width to zero:

```
UiObject slideOffDialogBox;
HliUnsignedShort border_width = 0;
UiSetBorderWidth (slideOffDialogBox, border_width);
```

Notes

Setting the border width to zero hides the dialog box as soon as the cursor is dragged outside the limits of the SlideOffDialogBox. If a border is present the dialog box would remain visible even if the cursor is dragged out of the box but is within the limits of the border.

UiSet...Cb

Purpose

Set a pointer to a callback procedure for an object. This allows you to substitute your own procedure for the one provided by CVact.

Syntax

```
HliStatus UiSetCallBackNameCb(
UiObject obj,
UiCallbackProc cb,
void *cd)
```

Description

The callback procedures can all be obtained by using the names in the following table in `UISet...Cb` function shown.

Table A-3 Callback Names

Name	Name
AbortCb	EditorFocusChangeCb
ActivateCb	EditorInvokedCb
ApplyCb	EditorRescindCb
CancelCb	Entry*n*Cb [see Note]
ChosenCb	EntryCb
ClosedCb	InitializeCb
CompleteCb	InputCb
CompleteCellCb	IsDropOkCb
ConfirmCb	IsEditableCb
DeactivateCb	IsInputOkCb
DefaultActionCb	MouseEventCb
DestroyCb	PropertiesCb
DismissCb	PushedCb
DropCb	ResetCb

Notes

For apply pulldowns, the following call is also available to set the callback procedure `proc` and `data` for entry `i` in Apply Pulldown obj.

```
HliStatus UiSetEntryCb
 ((UiObject obj,
  HliSignedShort i,
  UiCallbackProc proc,
  void *data));
```

UiSetCell

Purpose

Set the value of a cell in a scroll matrix.

Syntax

```
HliStatus UiSetCell (
UiObject obj,
HliSignedShort row,
HliSignedShort column,
HliConstString value)
```

Description

Set the value of a scroll matrix cell at row, column to the specified value.

Example

The following example sets the first cell to contain the value “Hello.”

```
UiObject ScrollMatrix;

/* Initialize the matrix using UiSetMatrix */


/* Assign the value of the first cell */
UiSetCell(ScrollMatrix, 0, 0, HliFromCString(”Hello”));
```

Notes

To set many cells, it may be more efficient to do a set on the Cells attribute.

UiSetCellColor

Purpose

Set the color value of a cell in a scroll matrix.

Syntax

```
HliStatus UiSetCellColor (
UiObject obj,
HliSignedShort row,
HliSignedShort column,
UiColor color)
```

Description

Set the color value to color for the cell at position row, column.

Notes

To set many cells, it may be more efficient to set the Color attribute.

UiSetCellValues

Purpose

Assign an array of values to an array of cells in a scroll matrix.

Syntax

```
HliStatus UiSetCellValues (
UiObject obj,
HliString *const cellValues[ ],
HliSignedShort rows,
HliSignedShort columns)
```

Description

Fill cells in a matrix where `cellValues` is an array of their contents, and rows and columns together indicate how many rows and columns to fill.

Example

The following example assumes that the scroll matrix has been initialized—see `UiSetMatrix`. It re-uses the cell values for the second row of the matrix.

```
UiObject ScrollMatrix;
HliString *SM1cells[3];

/* Initialize the cells */
SM1cells[0] = HliFromCString(”00”);
SM1cells[1] = HliFromCString(”01”);
SM1cells[2] = HliFromCString(”02”);

/* Fill the matrix (cells) with data *\
UiSetCellValues(ScrollMatrix, SM1cells, 3, 2);
```

Notes

Data from cellValues will be used to populate the scroll matrix in row-major order (left to right, top to bottom).

UiSetChosen

Purpose

Choose an item in an option list.

Syntax

```
HliStatus UiSetChosen (
UiObject obj,
HliSignedShort idx,
HliBoolean chosen)
```

Description

A chosen value of TRUE means make the item at position `idx` chosen. FALSE makes it not chosen, then making an item chosen makes another item not chosen (if there is a previously chosen item) and making an item not chosen is illegal.

Example

```
UiObject    Obj ;
HliSignedShort def = 0 ;
. . .
UiSetChosen( Obj, def, TRUE );
```

Notes

If RadioBehavior is set, making an item chosen makes another item not chosen (because only one item can be chosen at a time), and setting the chosen value to FALSE is illegal.

UiSetColumnAlignments

Purpose

Set the alignment information for columns in a matrix.

Syntax

```
HliStatus UiSetColumnAlignments (
  UiObject obj,
  UiAlignment *columnAlignments,
  HliSignedShort num)
```

Description

Set `num` columns to the alignments the array `columnAlignments`.

Example

The following example initializes a two-element array that right-justifies the first column's contents and centers the second column's contents.

```
UiObject ScrollMatrix
UiAlignment SM1colAlignments[2];

/* initialize the column alignments */
SM1colAlignments[0] = UI_END_ALIGN;
SM1colAlignments[1] = UI_CENTER_ALIGN;

UiSetColumnAlignments(ScrollMatrix, SM1colAlignments, 2);
```

UiSetColumnColors

Purpose

Set the color values of an entire column in a scroll matrix.

Syntax

```
HliStatus UiSetColumnColors (
Uiobject obj,
HliSignedShort position,

UIColor *colors,
HliSignedShort num_colors)
```

Description

Position indicates the column whose colors are to change. `Num_colors` indicates the count of colors in the array colors.

UiSetColumnLabel

Purpose

Sets the label for the specified column in a scroll matrix.

Syntax

```
HliStatus UiSetColumnLabel (
UiObject obj,
HliShortString columnLabel,
HliSignedShort idx)
```

Description

Set the `idx`'th column label to `columnLabel`.

UiSetColumnLabels

Purpose

Sets the labels for the specified column in a scroll matrix.

Syntax

```
HliStatus UiSetColumnLabel (s
UiObject obj,

HliString *columnLabels,
HliSignedShort num)
```

Description

Set `num` columns to the labels in the array `columnLabels`.

Example

The following example initializes a two-element array with the titles “Col 1” and “Col 2,” then uses `UiSetColumnLabels` with a pointer to the beginning of the array containing the labels.

```
UiObject ScrollMatrix;
HliString SM1colLabels[2];

/* initialize the column labels */
SM1colLabels[0] = HliFromCString(”Col 1”);
SM1colLabels[1] = HliFromCString(”Col 2”);

UiSetColumnLabels(ScrollMatrix, SM1colLabels, 2);
```

The following example assumes the same initialization as above, but sends UiSetColumnLabels a pointer to an array of pointers.

```
HliShortString *ptrSM1colLabels[2];

ptrSM1colLabels[0] =
(HliShortString *)HliFromCString(”COLUMN 1”);
ptrSM1colLabels[1] =
(HliShortString *)HliFromCString(”COLUMN 2”);

UiSetColumnLabels(ScrollMatrix,
(HliString *)ptrSM1colLabels, 2);
```

UiSetDefaultButtonWithSThickness

Purpose

Sets the default button on a dialog box with the specified shadow thickness in pixels.

Syntax

```
HliStatus UiSetDefaultButtonWithSThickness
(UiObject obj,
UiObject db,
HliUnsignedShort sthickness);
```

Sets the `DefaultButton` attribute of the dialog box to the specified Pushbutton object and sets the shadow thickness for the default button.

When the default button has focus, a shadow is drawn around it with the shadow thickess as specified to the `UiSetDefaultButtonWithSThickness` binding.

UiSetDependency

Syntax

```
HliStatus UiSetDependency (
UiObject obj,
UiDependency type,
const char *expression)
```

See Also

UIGet...

UiSetEditorAttributes

Syntax

```
HliStatus UiSetEditorAttributes (
UiObject obj,
UiIcon ico,
UiCallbackProc invokeProc,
void *invokeData,
UiCallbackProc rescindProc,
void *rescindData,
UiCallbackProc focusProc,
void *focusData)
```

See Also

UIGet...

UiSetEntryLabel

Purpose

Set an entry label in an apply pulldown.

Syntax

```
HliStatus UiSetEntryLabel
 ((UiObject obj,
  HliSignedShort i,
  HliConstShortString buf));
```

Description

Set the label string to `buf` for entry `i` in Apply Pulldown obj.

UiSetEntryRequired

Purpose

Set the required flag for an entry in a pulldown.

Syntax

```
HliStatus UiSetEntryRequired
 ((UiObject obj,
  HliSignedShort i,
  HliBoolean required));
```

Description

Set the required flag for the specified entry `i` in Apply Pulldown obj.

UiSet...FromMessage UiSet...FromVMessage

Purpose

This “family” of calls sets various textual objects from messages or a list of arguments.

Syntax

```
HliStatus UiSetTextualObjectFromMessage(
UiObject obj,
HliStatus message,
...)

HliStatus UiSetTextualObjectFromVMessage(
UiObject obj,
HliStatus message,
va_list args )
```

Textual Objects Set from Messages

The following is a list of textual objects which you can set from messages:

- ApplyLabel
- AssistLine
- DefaultString
- IconifiedTitle
- OKLabel
- Prompt
- TextString
- TextValue
- Title

UiSetGeometrySpecified

Purpose

Set the size and position of an object.

Syntax

```
HliStatus UiSetGeometrySpecified (
UiObject obj,
UiDimension x,
UiDimension y,
UiDimension width,
UiDimension height )
```

Description

Set the user specified x and y position, relative to its parent, and the user specified width and height for the UiObject obj to x, y, width, and height respectively.

See Also

UIGet...

UiSetInstanceId

Purpose

Sets the instance id of the specified UiObject which is the instance of a specified dialog or property sheet. You can use `UiSetInstanceId` to set the instance id for each of the multiple instances of a CVact menu.

Syntax

```
HliStatus UiSetInstanceId
(UiObject obj,
HliUnSignedShort inst_id);
```

Description

The Cl function `ClRegisterDialogMaxInstances` allows the creation of multiple instances of the specified property sheet or dialog box. The maximum number of instances allowed for a property sheet or dialog box is in the range 1 to 65,535.

The instance id is in the range 1 to 65,535. The Cl Library automatically sets the appropriate instance id on each of the multiple instances of the dialog box or property sheet, launched either from the CADDS algorithms using `cl_launch_dialog` or from the CR side using the `ClLaunchDialog` binding. In a session, the instance id and the assigned instance id remain the same and do not change.

Please note: Do not use `UiSetInstanceId` to assign an id to an instance of a property sheet or dialog box if you have used `cl_launch_dialog` or `ClLaunchDialog` to launch the dialog box or property sheet, already registered with `ClRegisterDialog`.

See Also

UiGetInstanceId

UiSetItemSensitive

Purpose

Sets the specified option list item sensitive.

Syntax

```
HliStatus UiSetItemSensitive
(UiObject obj, HliSignedShort Pos
HliBoolean State
);
```

Description

Displays an object normally or grays it out.

TRUE displays the item normally, allows it to obtain focus, and permits user input (if appropriate for this object). FALSE grays or hazes out the item, prevents user input, and disallows focus.

UiSetMatrix

Purpose

Initialize or reconfigure a scroll matrix.

Syntax

```
HliStatus UiSetMatrix (
UiObject obj,
HliSignedShort columns,
HliSignedShort rows,
HliString *cellValues[ ],
HliString *columnLabels,
HliString *rowLabels,

UiAlignment *alignments,
HliSignedShort *columnWidths)
```

Description

Set the RowsNum and columnsNum attributes for a scroll matrix to the values given by rows and columns. The labels for the rows and columns are specified by the arrays of `rowLabels` and `columnLabels`, and the label alignments by alignments. Column widths are given by `columnWidths`.

Example

The following example defines the components of a scroll matrix but assumes their initialization. It then calls `UiSetMatrix` to create a matrix having those (initialized) components.

```
#define SM2COLS 5
#define SM2ROWS 5

UiObject ScrollMatrix;
HliString *SM2cells[SM2ROWS];
UiAlignment SM2colAlignments[SM2COLS];
HliString SM2colLabels[SM2COLS];
HliString SM2rowLabels[SM2ROWS];
HliSignedShort SM2colWidths[SM2COLS];

/* initialize column alignments, widths, labels, etc. */
. . .

/* initialize cell contents */
. . .

UiSetMatrix(ScrollMatrix, SM2COLS, SM2ROWS, SM2cells,
```

```
SM2colLabels,SM2rowLabels,
SM2colAlignments, SM2colWidths);
```

Notes

Data from `cellValues` will be used to populate the scroll matrix in row-major order (left to right, top to bottom).

UiSetRowColor

Purpose

Set the colors of a row in a scroll matrix.

Syntax

```
HliStatus UiSetColumnColors (
Uiobject obj,
HliSignedShort position,
UIColor *colors,
HliSignedShort num_colors)
```

Description

Position indicates the row whose colors are to change. `Num_colors` indicates the count of colors in the array colors.

UiSetRowColumn

Syntax

```
HLI_EXT HliStatus UiSetRowColumn HLI_ARGS
    ((UiObject obj,
      HliSignedShort col,
      HliSignedShort wide,

      HliSignedShort row,
      HliSignedShort high));
```

UiSetRowLabel

Purpose

Set the label for the specified row in a scroll matrix.

Syntax

```
HliStatus UiSetRowLabel (
UiObject obj,
HliShortString rowLabel,
HliSignedShort idx)
```

Description

Set the label of the row at `idx` to `rowLabel`.

UiSetRowLabels

Purpose

Set the labels for the specified number of rows in a scroll matrix.

Syntax

```
HliStatus UiSetRowLabels (
UiObject obj,
HliShortString *rowLabels,
HliSignedShort num)
```

Description

Set `num` rows to the null-terminated labels found in the array `rowLabels`.

Example

The following example initializes a three-element array with the titles “Row 1” and so on, then uses `UiSetRowLabels` with a pointer to the beginning of the array containing the labels.

```
UiObject ScrollMatrix
HliString SM1rowLabels[3];

/* initialize the row labels */
SM1rowLabels[0] = HliFromCString(”Row 0”);
SM1rowLabels[1] = HliFromCString(”Row 1”);
SM1rowLabels[2] = HliFromCString(”Row 2”);

UiSetRowLabels(ScrollMatrix, SM1rowLabels, 3);
```

UiSlideOffDialogBoxCreate

Purpose

Create a dialog box with slide-off behavior.

Syntax

```
HliStatus UiSlideOffDialogBoxCreate (
UiObject *obj,
const char *name,
UiHelpContext help,
HliConstShortString winTitle,
UiIcon iconIcon,
HliConstShortString iconTitle,
HliBoolean modal)
```

Description

This is a convenience function for creating a DialogBox with slide-off behavior.

UiSlideOffEditorDialogBoxCreate

Syntax

```
HliStatus UiSlideOffEditorDialogBoxCreate (
UiObject *obj,
char *name,
HliString winTitle,
UiHelpContext help,
UiIcon iconIcon,
HliConstShortString iconTitle,
HliBoolean modal)
```

UiSetSpecifiedPosition

Purpose

Set the user-specified position of an object.

Syntax

```
HliStatus UiSetSpecifiedPosition
  ((UiObject obj,
   UiDimension x,
   UiDimension y));
```

Description

Set the user specified x and y position, relative to its parent, for the UiObject obj to x and y.

UiSetSpecifiedSize

Purpose

Set the user-specified size of an object.

Syntax

```
HliStatus UiSetSpecifiedSize
  ((UiObject obj,
   UiDimension w,
   UiDimension h));
```

Description

Set the user specified width and height for the UiObject obj to width and height respectively.

UiSpecifierCreate

Purpose

Used to display a specifier in the specifier bar.

Syntax

```
HliStatus UiSpecifierCreate (
UiObject *obj,
const char *name,
UiHelpContext help,
HliConstShortString assist,
UiKey accelerator,
HliConstShortString textValue,
UiIcon icon,
UiIcon alternateIcon,
UiCallbackProc activateCb,
void *activateCd,
HliBoolean checked)
```

UiStartBusy

Purpose

Display an hourglass.

Syntax

```
HliStatus UiStartBusy
```

Description

Display an hourglass to show that something is happening.

Example

```
HliStatus status;
. . .
status = UiStartBusy ;
```

See Also

UiEndBusy

UiToggleCreate

Purpose

Create a toggle button.

Syntax

```
HliStatus UiToggleCreate (
UiObject obj
HLI_CONST char *name,
UiObject parent,
HliConstShortString assist,
UiHelpContext help,
HliConstShortString label,
UiIcon icon,
UiIcon altIcon,
UiCallbackProc chosenCb,
void *chosenCd,
HliBoolean defaultValue)
```

Description

Creates a toggle and assigns it to obj.

Example

```
HliStatus status;
UiObject Toggle1, RadioSet1;
. . .
/* first, create the radio set to hold the toggle button */
. . .
status = UiToggleCreate( &Toggle1, "Toggle1", RadioSet1
/* AssistLine */ ,HLI_NULL_STRING
/* HelpContext */ ,UI_NULL_HELP
/* TextValue */ ,HLI_NULL_STRING
/* Icon */ ,UI_NULL_ICON
/* AlternateIcon */ ,UI_NULL_ICON
/* ChosenCb */ ,UI_NULL_CALLBACK
 ,NULL
/* DefaultBoolean */ ,FALSE
);
```

UiToggleEntryCreate

Purpose

Used as a special case of direct action entries which set a boolean state in the application.

Syntax

```
HliStatus UiToggleEntryCreate (
UiObject *obj,
const char *name,
UiHelpContext help,
HliConstShortString assist,
UiKey accelerator,
HliConstShortString textValue,
UiIcon icon,
UiIcon alternateIcon,
UiCallbackProc activateCb,
void *activateCd,
HliBoolean checked)
```

Description

Creates a toggle entry and assigns it to obj.

Notes

The system data is the new value of Checked. System data is for this callback is UiChosenStruct.

UiWarningBoxCreate
UiWarningBoxCreateFromStatus
UiWarningBoxCreateFromVStatus

Purpose

To display warning messages to the user, such as, “Are you sure you want to delete this part.”

Syntax

```
HliStatus UiWarningBoxCreate (
UiObject *obj,
char *name,
UiHelpContext help,
UiIcon iconIcon,
HliConstShortString iconTitle,
UiCallbackProc okCb,
void *okData,
UiCallbackProc cancelCb,
void *cancelData,

/* UiWarningBoxCreate */
HliString text_string)

/* UiWarningBoxCreateFromStatus */
HliStatus status,
. . .)


/* UiWarningBoxCreateFromVStatus */
HliStatus status,
va_list args)
```

Description

For `UiWarningBoxCreate`, the message is taken from the HliString passed in as `text_string`. The warning dialog box is then created displaying the contents of the HliString.

UiWorkingBoxCreate

Purpose

Creates a WorkingBox, used to show that a task or operation is in progress.

Syntax

```
HliStatus UiWorkingBoxCreate (
UiObject *obj,
char *name,
UiHelpContext help,
UiIcon iconIcon,
HliConstShortString iconTitle,
HliBoolean hascancelPb,
UiCallbackProc okCb,
void *okData,
UiCallbackProc cancelCb,
void *cancelData,
HliString text_string )
```

Appendix B

Hli Functions

The Hli functions were designed primarily for purposes of internationalization. They provide a set of string- and character-handling routines that create and modify text so that it can be stored in external files (for example, message repositories), translated, inserted into CVact objects, and so on.

The Hli header files define types and functions for retrieving, manipulating, and displaying user-visible text. To support internationalization, there are separate types for internal and external strings since they have different representations on different platforms. (Internal strings are those used within the application for manipulation, typically with Ui functions; external strings are those used in files.)

For this release, the descriptions of some of the Hli functions are not complete. This affects the book only, and not the Hli functions themselves.

- HliCCharToChar ()
- HliCNStringToString ()
- HliCStrdup ()
- HliCStringFromString ()
- HliCStringToString ()
- HliCStrndup ()
- HliCharToCChar ()
- HliExternNStringToString ()
- HliExternStringFromString ()
- HliExternStringToString ()
- HliFromCString ()
- HliFromExternString ()
- HliShutdown ()

- HliStringNToCString ()
- HliStringNToExternString ()
- HliStringToCString ()
- HliStringToExternString ()
- HliStringVFormat ()

HliCCharToChar ()

Syntax

```
HLI_EXT HliStatus HliCCharToChar HLI_ARGS (
CChar from,
HliChar *to)
```

Description

If the `from` character, assumed to be a byte in the seven-bit ASCII encoding, is representable as an HliChar, it is converted to the HliChar encoding; otherwise, the result is a space and an error is returned.

Example

```
CChar c = 'a';
HliChar hli_c;
HliCCharToChar(c, hli_c);
```

HliCNStringToString ()

Syntax

```
HLI_EXT HliStatus HliCNStringToString HLI_ARGS (
HLI_CONST char* from,
HliSignedLong fromLen,
HliString to,
HliSignedLong toLen)
```

Description

Convert and copy `fromLen` chars from a C string to `toLen` chars in an HliString, where `from` is a C array or C string and `to` is of type HliString or HliShortString.

Example

```
char from[256];
HliShortString to;
HliSignedLong n_c = 3;
HlisignedLong n_hli = 3;
strcpy(from, "Now is the time");
/* Put "Now" into the string "to" */
HliCNStringToString (from,n_c, to, n_hli);
```

Notes

HliCNStringToString will convert `fromLen` characters or up to terminating NULL char ('\0') - which ever comes first from the C string. It will copy `toLen`-converted HliChars into the HliString or return HLI_FAIL on error. The terminating HLI_NULL_CHAR is placed in the `to` string for you.

HliCStrdup ()

Purpose

HliCStrdup allocates memory for and then copies one C string to the newly allocated C string.

Syntax

```
HLI_EXT CString HliCStrdup HLI_ARGS (
HLI_CONST CChar* from)
```

Description

`from` **--**HliCStrdup allocates memory for and then copies one C string to the newly allocated C string.

Example

```
char *previously_created_string;
char *new_string;
new_string = HliCStrdup(previously_created_string);
if(new_string == NULL)
         /* then HliMalloc failed to allocate space */
```

Notes

This function returns dynamically allocated memory using HliMalloc. It is the responsibility of the application or application programmer to manage the allocated memory.

It is not necessary to declare the C string (CChar) as HLI_CONST. It is declared as constant in the function prototype simply to protect the string being passed in.

See Also

HliMalloc_()
HliFree_()

HliCStringFromString ()

Syntax

```
HLI_EXT CString HliCStringFromString HLI_ARGS (
HLI_CONST HliChar* from)
```

Description

Converts and copies contents of HliString, “from”, into a newly allocated C string.

Example

```
HliString prev_created_hli_string;
char *new_cs;
new_cs = HliCStringFromString(prev_created_hli_string);
if(new_cs == NULL)
   /* problem allocating storage */
```

Notes

This function returns a C string for which HliMalloc_ has allocated memory. The application or application programmer is now responsible for management of this memory.

See Also

HliMalloc_()
HliFree_()

HliCStringToString ()

Syntax

```
HLI_EXT HliStatus HliCStringToString HLI_ARGS (
HLI_CONST CChar* from,
HliString to)
```

Example

```
CChar prev_created_string;
HliString new_hli_s;
int len;
len = strlen(prev_created_string);
new_hli_s = (HliString) HliMalloc_(((len + 1) * sizeof(HliChar));
```

Notes

This function returns a HliString for which `HliMalloc_` has allocated memory. The application or application programmer is now responsible for management of this memory.

See Also

HliMalloc_()
HliFree_()

HliCStrndup ()

Syntax

```
HLI_EXT CString HliCStrndup HLI_ARGS (
HLI_CONST CChar* from,
HLI_CONST HliSignedLong len)
```

HliCharToCChar ()

Syntax

```
HLI_EXT HliStatus HliCharToCChar HLI_ARGS (
HliChar from,
CChar *to)
```

HliExternNStringToString ()

Purpose

Internationalization

Syntax

```
HLI_EXT HliStatus HliExternNStringToString HLI_ARGS (
HLI_CONST HliExternChar* from,
HliSignedLong fromLen,
HliString to,
HliSignedLong toLen)
```

Description

The external file character set encoding is represented by this string type. Functions are provided for converting file data to and from HliStrings. Any inconvertible characters become spaces and cause warning status to be returned.

In Unix, a distinction is made between internal wide characters and external multi-byte characters. Other platforms may also make a distinction between the character representation used internally and in files, so applications should make no assumptions about representations. The functions below provide translation between the external file encoding and the internal process encoding. ANSI C provides this functionality with functions named `wcstombs` and `mbstowcs`.

Notes

This function returns a string for which `HliMalloc_` has allocated memory and which the application must `HliFree_` .

HliExternStringFromString ()

Purpose

Internationalization

Syntax

```
HLI_EXT HliExternString HliExternStringFromString HLI_ARGS (
HLI_CONST HliChar* from)
```

Description

The external file character set encoding is represented by this string type. Functions are provided for converting file data to and from HliStrings. Any inconvertible characters become spaces and cause warning status to be returned.

In Unix, a distinction is made between internal wide characters and external multi-byte characters. Other platforms may also make a distinction between the character representation used internally and in files, so applications should make no assumptions about representations. The functions below provide translation between the external file encoding and the internal process encoding. ANSI C provides this functionality with functions named `wcstombs` and `mbstowcs`.

Notes

This function returns a string for which `HliMalloc_` has allocated memory and which the application must `HliFree_` .

HliExternStringToString ()

Purpose

Internationalization

Syntax

```
HLI_EXT HliStatus HliExternStringToString HLI_ARGS (
HLI_CONST HliExternChar* from,
HliString to)
```

Description

The external file character set encoding is represented by this string type. Functions are provided for converting file data to and from HliStrings. Any inconvertible characters become spaces and cause warning status to be returned.

In Unix, a distinction is made between internal wide characters and external multi-byte characters. Other platforms may also make a distinction between the character representation used internally and in files, so applications should make no assumptions about representations. The functions below provide translation between the external file encoding and the internal process encoding. ANSI C provides this functionality with functions named `wcstombs` and `mbstowcs`.

Notes

This function returns a string for which `HliMalloc_` has allocated memory and which the application must `HliFree_` .

HliFromCString ()

Syntax

```
HLI_EXT HliString HliFromCString HLI_ARGS (
HLI_CONST CChar* from)
```

Notes

This function returns a string for which `HliMalloc_` has allocated memory and which the application must `HliFree_` .

HliFromExternString ()

Purpose

Internationalization

Syntax

```
HLI_EXT HliString HliFromExternString HLI_ARGS (
HLI_CONST HliExternChar* from)
```

Description

The external file character set encoding is represented by this string type. Functions are provided for converting file data to and from HliStrings. Any inconvertible characters become spaces and cause warning status to be returned.

In Unix, a distinction is made between internal wide characters and external multi-byte characters. Other platforms may also make a distinction between the character representation used internally and in files, so applications should make no assumptions about representations. The extern string functions translate between the encodings for external files and internal processes. ANSI C provides this functionality with functions named `wcstombs` and `mbstowcs`.

Notes

This function returns a string for which `HliMalloc_` has allocated memory and which the application must `HliFree_` .

HliShutdown ()

Syntax

```
HLI_EXT HliStatus HliShutdown HLI_ARGS (
void)
```

HliStringNToCString ()

Purpose

Internationalization

Syntax

```
HLI_EXT HliStatus HliStringNToCString HLI_ARGS (
HLI_CONST HliChar* from,
HliSignedLong fromLen,
CString to,
HliSignedLong toLen)
```

Description

The external file character set encoding is represented by this string type. Functions are provided for converting file data to and from HliStrings. Any inconvertible characters become spaces and cause warning status to be returned.

In Unix, a distinction is made between internal wide characters and external multi-byte characters. Other platforms may also make a distinction between the character representation used internally and in files, so applications should make no assumptions about representations. The functions below provide translation between the external file encoding and the internal process encoding. ANSI C provides this functionality with functions named `wcstombs` and `mbstowcs`.

Notes

This function returns a string for which `HliMalloc_` has allocated memory and which the application must `HliFree_` .

HliStringNToExternString ()

Purpose

Internationalization

Syntax

```
HLI_EXT HliStatus HliStringNToExternString HLI_ARGS (
HLI_CONST HliChar* from,
HliSignedLong fromLen,
HliExternString to,
HliSignedLong toLen)
```

Description

The external file character set encoding is represented by this string type. Functions are provided for converting file data to and from HliStrings. Any inconvertible characters become spaces and cause warning status to be returned.

In Unix, a distinction is made between internal wide characters and external multi-byte characters. Other platforms may also make a distinction between the character representation used internally and in files, so applications should make no assumptions about representations. The functions below provide translation between the external file encoding and the internal process encoding. ANSI C provides this functionality with functions named `wcstombs` and `mbstowcs`.

HliStringToCString ()

Syntax

```
HLI_EXT HliStatus HliStringToCString HLI_ARGS (
HLI_CONST HliChar* from,
CString to)
```

HliStringToExternString ()

Purpose

Internationalization

Syntax

```
HLI_EXT HliStatus HliStringToExternString HLI_ARGS (
HLI_CONST HliChar* from,
HliExternString to)
```

Description

The external file character set encoding is represented by this string type. Functions are provided for converting file data to and from HliStrings. Any inconvertible characters become spaces and cause warning status to be returned.

In Unix, a distinction is made between internal wide characters and external multi-byte characters. Other platforms may also make a distinction between the character representation used internally and in files, so applications should make no assumptions about representations. The functions below provide translation between the external file encoding and the internal process encoding. ANSI C provides this functionality with functions named `wcstombs` and `mbstowcs`.

HliStringVFormat ()

Syntax

```
HLI_EXT HliStatus HliStringVFormat HLI_ARGS (
HliString result,
HLI_CONST HliChar* format,
va_list args)
```

Appendix C

CL Functions

The CL functions communicate with CADDS. They send commands to CADDS' text input stream as if the user had entered them there. They interrogate CADDS to see if it is ready to accept such commands.

These functions are declared in `clwrap.h`. They let you send a command to CADDS and fetch runtime data from CADDS. They also allow CADDS to invoke a UIHLI sheet and CR to access CADDS information.

This appendix helps you to look up the syntax and signatures of the functions.

- Notes on CL Functions
- ClAbortCommand
- ClAssistOn
- ClAVAddCallback
- ClAVCreate
- ClAVGetDoubleVal
- ClAVGetFloatVal
- ClAVGetHliStringVal
- ClAVGetIntegerVal
- ClAVGetStringVal
- ClAVGetValue
- ClAVRemoveCallback
- ClAVSetDoubleVal
- ClAVSetFloatVal
- ClAVSetHliStringVal
- ClAVSetIntegerVal

- ClAVSetStringVal
- ClAVSetValue
- ClBufferCCommand
- ClBufferMsg
- ClCaddsRtlInstanceWrap
- ClCaddsRtlSizeWrap
- Cl_call_CR_func
- ClClearPrompt
- Cl_E_epsilon_value
- Cl_E_extents_value
- ClFlushCommand
- ClGetDialogInstances
- ClGetDialogLastInstanceId
- ClGetDialogMaxInstances
- ClGetDialogNumCurInstances
- ClGetFloatStringVal
- ClGetIntegerStringVal
- ClInitialize
- ClLaunchDialog
- ClRegisterDialog
- ClRegisterDialogMaxInstances
- ClRequest
- Cl_RTL_caddspaths_size
- Cl_RTL_e_drawing_name_init
- Cl_RTL_e_drawing_name_instance
- Cl_RTL_e_drawing_name_size
- Cl_RTL_e_nfig_all_size
- Cl_RTL_e_tag_all_size
- Cl_RTL_e_view_name_size
- Cl_RTL_caddspaths_get_name
- ClSendCCommand
- ClSendErrorCode

- ClSendErrorCString
- ClSendErrorHliString
- ClSendWarning
- ClSetPrompt

Notes on CL Functions

Sending Commands to CADDS

The following are the functions for sending a command to CADDS

- ClSendCCommand(CChar *format, ...)
- ClSendComand(HliChar *format, ...)
- ClBufferCCommand(const CChar *format, ...)
- ClBufferCommand(HliChar *format, ...)
- ClBufferMsg(msgId)
- ClFlushCommand()
- ClAbortCommand()

`ClSendCCommand` and `ClSendComand` send the command to CADDS right away.

`ClBufferCCommand`, `ClBufferCommandn` and `ClBufferMsg` concatenate together the command fragments into one long command, which will be sent later. They have the advantage of letting you analyze piece by piece the data that you have to send, without having to manage the memory allocation for the command fragments yourself.

`ClAbortCommand` tells the system to throw away all of the buffered commands that you may have stored. This is useful if, when analyzing the data to send to CADDS, the program discovers a user error. At this point, the program decides that the command should not be sent off at all.

`ClFlushCommand` sends to CADDS all of the buffered command information that you may have stored with `ClBufferCCommand`, `ClBufferCommand`, and `ClBufferMsg`. `ClSendCCommand` and `ClSendComand` will also have the affect of sending the previously buffered information to CADDS.

The input to `ClBufferMsg` would be one of the symbol constants defined in a `.h` file generated from a UIHLI `.mrs` file.

The format statement in some of these is a `printf` style format statement. However, the only format directives allowed currently are %S and %s. %S means one of the arguments is an HliString. %s means that one of the arguments is a C String.

`ClSendCCommand` allows you to just put together format statements, such as ”%S %S %s”. You are likely to use this function if you have a quantity of HliString variables containing the pieces of your command to CADDS.

`ClSendCommand` requires that the format string be an HliString. This would be useful if instead you used the function HliStrcat to put together the components of your command, before calling `ClSendCommand`. In this case, you’d just have an HliChar *format, and wouldn’t have any arguments.

`ClBufferCommand` and `ClBufferCCommand` have the same rules for format statements as `ClSendCommand` and `ClSendCCommand`. These commands save you the trouble of doing `HliMalloc_` and `HliStrcat` yourself.

Invoking a CR Function from CADDS

You can directly invoke a CR function from CADDS. A generic interprocess communication utility is provided for this ease of communication between CADDS and CR. CR is the composer replacement process which manages all CVact menu requirements of CADDS.

This communication interface enables you to design interactive menus like New NURBS and Sketcher of the CADDS environment for highly interactive applications.

This utility reduces the complexity of interprocess communication between CADDS and CR. Earlier the interprocess communication between CADDS and CR was restricted or limited by a request for launching a specified CVact menu. CR only launched the requested menu.

The call to CR function blocks CADDS, that is, CADDS is blocked until the CR process is complete. But CADDS may also send a request to CR function to complete the process and may not wait for a reply from CR.

Now you can launch a CVact menu from CADDS algorithms and also pass the field contents using a single cell. Functions in CR callable using the CADDS-CR communication interface is registered by a dynamic registration mechanism. The CADDS-CR communication interface is identical to the ClRequest interface. Use ClRequest to initialize the menu and retrieve information on the menu field contents. However, it differs by the following exceptions:

- The argument mode CL_MODEoINREF does not exist
- Arrays of basic data types are not provided

- String buffer basic data type which is the preallocated array of characters is also not provided

The description of the function is as follows:

- Return data types : `void, integer, float, double, string`
- Argument data types : `integer, float, double string`
- Function argument passing can follow any of the following conventions : pass by value (`CL_MODE_IN`), pass by reference with only return value expected (`CL_MODE_OUT`) and pass by reference with the intention of passing a value and expecting a modified return value (`CL_MODE_INOUT`).

Public Interfaces to CADDS Functions

The CL interface includes support for several wrapper functions. These wrapper functions include support for the following

- Fetching common CADDS globals
- Initialization, size and instance functions for RTL extractions.
- Calling what was formerly called composer extraction functions. These are functions with no argument that only return a single value.

The list of these supplied wrapper functions is documented in the include file `clwrap.h`.

If there are any RTL extraction functions that you need access to, but are not declared in `clwrap.h`, it is easy to add them very quickly yourself. We have supplied three convenience functions which are also declared in `clwrap.h`. In all of these, `func` is the name of the particular RTL extraction routine in CADDS. The following lists these functions:

- Convenience function for accessing any function of type RTL_....._init

```
void ClCaddsRtlInitWrap(const char *func)
```

- Convenience function for accessing any function of type RTL_....._instance

```
 HliString ClCaddsRtlInstanceWrap(const char *func,HliSignedLong
index)
```

- Convenience function for accessing any function of type RTL_....._size

```
HliSignedLong ClCaddsRtlSizeWrap(const char *func)
```

There is also a convenience function for automatically initializing a UIHLI scrolllist object from a the CADDS RTL support function. This convenience function deletes any existing data in the scrolllist, and replaces it with the latest

up-to-date info from CADDS. It calls the three RTL convenience functions listed above. Here is the function definition

```
HliStatus CL_RTL_scrolllist (UiObject scrolllist,
      char *rtlName, HliBoolean callInit).
```

`clwrap.h` has a description of the arguments.

Please note: Given an rtlName `RTL_e_drawing_name`, this routine will try to call `RTL.e_drawing_name_init`, `..._size`, and `..._instance`. If there are no such functions, then an error message will appear in CADDS stating that there is no such foreign function.

Sending Coding Error Messages To CADDS

We have provided three interfaces for sending coding error messages to CADDS. The error messages will be displayed in the appropriate Explicit or Parametric port for such errors. When a coding error is encountered, in your own modules, the application developer has the option of using one of these interfaces for notifying CADDS

- ClSendErrorCode
- ClSendErrorCString
- ClSendErrorHliString

All of these are declared in Cl.h.

These should not be used for user errors. Users should be notified of user errors through a UIHLI Error or Warning Box (see `UiCreateErrorbox`, `UiCreateWarningBox`).

Using Active Variables between CADDS, CV-DORS, and CR Process

Active variables exist in the CR process and are shared between CADDS and the CR processes. The following points explain how the CADDS process and the CR process use the active variables:

- The CADDS process has only `WRITE` permissions on the active variables.
- The CR process has `READ/WRITE` permissions on the active variables.
- The CADDS process uses the appropriate CL library bindings to set the values of the active variables in the CR process.

- The CR process can get or set the values of the active variables using the appropriate CL bindings.
- You can fire a callback in CR when there is a change in the value of the active variable.
- Active variables are used for communicating the values of globals from CADDS to the CR process.

Each active variable is identified by a specific name or string. Specify the name of the active variable when creating the active variable on the CR side. CADDS specifies the active variable name when setting the value of an active variable.

Active Variable Types

Active variables are of the following types:

- `CL_TYPE_INTEGER` : integer type
- `CL_TYPE_FLOAT` : float type
- `CL_TYPE_DOUBLE` : double type
- `CL_TYPE_STRING` : string type

CL Bindings for Active Variables from CADDS

An active variable can be set in CADDS with any one of the following calls:

- `void cl_av_setvalue (char *name, ClType type, void *value)` Generic binding to set value of any type of active variable (of type `CL_TYPE_INTEGER`, `CL_TYPE_FLOAT`, `CL_TYPE_DOUBLE`, `CL_TYPE_STRING`)
- `void cl_av_set_integer_value (char *name, int value)` Binding to set value of an integer type (`CL_TYPE_INTEGER`) active variable
- `void cl_av_set_float_value (char *name, float value)` Binding to set value of a float type (`CL_TYPE_FLOAT`)active variable
- `void cl_av_set_double_value (char *name, double value)` Binding to set value of a double type (`CL_TYPE_DOUBLE`) active variable
- `void cl_av_set_string_value (char *name, char *value)` Binding to set value of a string type (`CL_TYPE_STRING`) active variable

Please note: The CADDS process can only set the value of the active variables. It cannot get the value of the active variables. CADDS must maintain its own globals in order to maintain the values of the active variables.

You can call these CL bindings from anywhere in CADDS algorithms. The implementation of these CL bindings from CADDS is contained in the `/cadds/lib/cue/cl/cl_av_set.c` file.

CL Bindings for Active Variables from CR

An active variable can be set in CR with any one of the following CL bindings that is documented in the following sections of this appendix:

- `HliStatus ClAVCreate (ClAVObject *av, ClType type, const char *name)` Create an active variable on specified type with specified name.
- `HliStatus ClAVAddCallback (ClAVObject av, ClAVCallback callback, void *data)` Add a callback on an active variable
- `int ClAVGetIntegerVal (ClAVObject av,)`
  Get value of an integer active variable
- `HliStatus ClAVSetIntegerVal (ClAVObject av, int value)`
  Set value of an integer active variable
- `float ClAVGetFloatVal (ClAVObject av)`
  Get value of a float active variable
- `HliStatus ClAVSetFloatVal (ClAVObject av, double value)`
  Set value of a float active variable
- `double ClAVGetDoubleVal (ClAVObject av)`
  Get value of a double active variable
- `HliStatus ClAVSetDoubleVal (ClAVObject av, double value)`
  Set value of a double active variable
- `HliString ClAVGetHliStringVal (ClAVObject av)`
  Get value of a string type active variable as a HliString
- `HliStatus ClAVGetHliStringVal (ClAVObject av, HliString value)` Set value of a string type active variable as a HliString
- `CString ClAVGetStringVal (ClAVObject av)`
  Get value of a string type active variable as a CString
- `HliStatus ClAVGetHliStringVal (ClAVObject av, CString value)` Set value of a string type active variable as a CString

The implementation of these CL bindings from CR is contained in the `/cadds/lib/cl/clav.cxx` file.

ClAbortCommand

Purpose

Tells the system to throw away all of the buffered commands that you may have stored.

Syntax

```
HliStatus ClAbortCommand()
```

Notes

This is useful if, when analyzing the data to send to cadds, the program discovers a user error. At this point, the program decides that the command shouldn't be sent off at all.

ClAssistOn

Purpose

In standalone mode, enables or disables echoing the Assist line to the standard out (or UIM/X message area).

Syntax

```
HliStatus ClAssistOn (HliBoolean state)
```

Example

```
ClAssistOn(FALSE);
```

Notes

TRUE and FALSE are HliBoolean values that are provided for you.

ClAVAddCallback

Purpose

Establish an interest in an active variable. Whenever the variable is modified, the registered callback is passed the specified data.

Syntax

```
HliStatus ClAVAddCallback
            (ClAVObject av,
            ClAVCallback callback,
            void *data)
```

Description

`av` is a handle returned with `ClAVCreate()`.

`callback` is the function to be called by `ClAVSetValue()`.

`data` is some data to be passed to the callback; an example would be a UiObject.

The example below defines and then adds a callback, `print_part_name,` to the active variable `CL_PART_STATE`, which is an active variable provided for you. Whenever a new part is opened or closed, the value of `CL_PART_STATE` changes and `print_part_name()` is called. If it changes because a part has been opened, its value is 1, otherwise 0.

Example

```
part_Name_Txt_Fillin; /* Created elsewhere */

/*Now define the function*/
void print_part_name(data)
      void *data;
{
HliString part_namep;
UiObject txt_fillin = (UiObject) data; /*cast*/

      part_namep = Cl_G_curr_part();
      if(ClAVGetIntegerVal(CL_PART_STATE == 1)) {
            UiSetTextString(txt_fillin,         part_namep);
            HliFree_(part_namep);


} /* end print_part_name() */
```

```
/*Register the function so it is called back*/
ClAVAddCallback(CL_PART_STATE, print_part_name,
part_Name_Txt_Fillin);
```

Notes

More than one function can be added as a callback to an active variable. They are called in the order added—the first one added is the first one called and so on.

`CL_PART_STATE` is an active variable of `CL_INTEGER_TYPE`**,** which is available as part of the CL library functionality, so it is not necessary to create.

`Cl_G_curr_part()` (also from the CL library) dynamically allocates space for the HliString, so you should free up `part_namep` after use.

See Also

ClAVSetValue() ()
ClAVRemoveCallback ()

ClAVCreate

Purpose

Create and register an active variable.

Syntax

```
HliStatus ClAVCreate()
            (ClAVObject *av
            ClType type,
            const char *name)
```

Description

`av` is the returned handle to the active variable created by this function.

`type` is one of `CL_TYPE_INTEGER`, `CL_TYPE_STRING`, `CL_TYPE_FLOAT`, or `CL_TYPE_DOUBLE`.

`name` is the name of the active variable to register.

Example

```
ClAVObject num_parts_modified;
      ClAVCreate(&num_parts_modified, CL_TYPE_INTEGER,
                        "num_parts_modified");
```

Notes

It is customary, but not required, to make the literal name the same as the variable name.

Active variables must be created before they can be operated on. One creates an active variable by calling ClAVCreate. After creating an active variable, it may have the following operations performed on it Set, Get, Add Callback.

All of the interfaces for active variables in the CR process are defined in the include file `ClAV.h`. You must include `Cl.h` in all of your independently created source files in order to get `ClAV.h` included also.

All creation of active variables should be done in your `init` function. This function is called at startup time. A sample template for this file is supplied in the examples directory (see aecinit.cxx).

See Also

ClAVSetValue()()
ClAVGetValue()
ClAVAddCallback()
ClAVRemoveCallback()

ClAVGetDoubleVal

Purpose

Get the value of an active variable of CL_TYPE_DOUBLE.

Syntax

```
double ClAVGetDoubleVal (ClAVObject av)
```

Description

`av` is an active variable of CL_TYPE_DOUBLE. Return value is the value of the active variable or 0.0 on error. Errors can be that the active variable doesn't exist, or that the active variable is not of type double. In the event of error, a message will be sent to the CADDS message buffer.

Example

```
double last_line_len;
HliDouble line_len;
ClAVObject curr_line_len;
      ClAVCreate(&curr_line_len, CL_TYPE_DOUBLE,
                        "curr_line_len");


      /* Code where curr_line_len is set to something */
      /*                  ...                          */
      last_line_len = clAVGetDoubleVal(curr_line_len);
```

Notes

What if 0.0 is a valid return value? Generally, the above error conditions should be eliminated in test mode or simple debugging so that a return value of 0.0 can be trusted as a valid number and not an error condition. Or, use `ClAVGetVal()`.

See Also

ClAVSetDoubleVal ()
ClAVGetVal ()

ClAVGetFloatVal

Purpose

Get the value of an active variable of type `CL_TYPE_FLOAT`.

Syntax

```
float ClAVGetFloatVal (ClAVObject av)
```

Description

`av` is an active variable of `CL_TYPE_FLOAT`. Return value is the value of the active variable or 0.0 on error. Errors can be that the active variable doesn't exist, or that the active variable is not of type double. In the event of error, a message will be sent to the CADDS message buffer.

Example

```
double tol;
HliDouble Tolerance;
ClAVObject curr_line_len;
      ClAVCreate(&Tolerance, CL_TYPE_DOUBLE,
                        "Tolerance");

      /* Code where Tolerance is set to something */
      /*                  ...                          */
      tol = ClAVGetFloatVal(curr_line_len);
```

Notes

The value to be set needs to be a double because K&R C passes floats as doubles.

What if 0.0 is a valid return value? Generally, the above error conditions should be eliminated in test mode or simple debugging so that a return value of 0.0 can be trusted as a valid number and not an error condition. Or, use `ClAVGetVal()`.

See Also

ClAVGetDoubleVal ()
ClAVSetFloatVal ()
ClAVGetValue ()

ClAVGetHliStringVal

Purpose

Get the value of an active variable of type `CL_TYPE_STRING`.

Syntax

```
HliString ClAVGetHliStringVal (ClAVObject av)
```

Description

`av` is an active variable of `CL_TYPE_STRING`. Return value is a transient HliString or `HLI_NULL_STRING` on error.

Example

```
ClAVObject view_name;
HliString hli_t;
HliShortString hli_tmp_s
      ClAVCreate(&view_name, CL_TYPE_STRING,
                        "view_name");

      /* Code where ClAVSetHliStringVal ( ) is used */
      /*                 ...                        */

      /* Later */
      hli_t = clAVGetHliStringVal(view_name);

/* Put transient HliString into something more permanent */
      if(hli_t != HLI_NULL_STRING) {
        HliStrcpy(hli_tmp_s, hli_t);
        /* Now, perhaps, set the TextString value of some */
        /* Fillin field with the value of hli_tmp_s.      */
     }
       else {
              ...
       }
```

Notes

`ClAVGetHliStringVal()` returns a transient pointer which is valid for a limited time. Therefore, before doing anything else, copy it into a more permanent variable. Transient pointers are automatically managed and should not be freed by the application.

See Also

ClAVSetHliStringVal ()
ClAVGetStringVal ()
ClAVGetVal ()
ClAVSetVal ()

ClAVGetIntegerVal

Purpose

Get the value of an active variable of type `CL_TYPE_INTEGER`.

Syntax

```
int ClAVGetIntegerVal (ClAVObject av)
```

Description

`av` is an active variable of `CL_TYPE_INTEGER`. Return value is the value of the active variable or 0 on error. Errors can be that the active variable doesn't exist, or that the active variable is not of type double. In the event of error, a message will be sent to the CADDS message buffer.

Example

```
ClAVObject AM;
      ClAVCreate((&AM, CL_TYPE_INTEGER,
                       "AM");

      /* Code where AM is set to 1 if time is morning */
      /* and 0 otherwise  ...                          */


      if( ClAVGetIntegerVal(AM) ) {
            printf("Good Morning ...");
      }
            else {
                  printf("Good Afternoon ...");
            }
```

Notes

What if 0 is a valid return value, as in the above example? Generally, the above error conditions should be eliminated in test mode or simple debugging so that a return value of 0 can be trusted as a valid number and not an error condition. Or, use `ClAVGetVal ()`.

See Also

ClAVSetIntegerVal ()
ClAVSetValue ()
ClAVGetValue ()

ClAVGetStringVal

Purpose

Get the value of an active variable of type CL_TYPE_STRING.

Syntax

```
CString ClAVGetStringVal (ClAVObject av)
```

Description

av is an active variable of CL_TYPE_STRING. Return value is a transient CString or (CString) NULL on error. Errors can be that the active variable doesn't exist, or that the active variable is not of type CL_TYPE_STRING In the event of error, a message will be sent to CADDS message buffer.

Example

```
char *c_tp;
char c_temp_part_notes_path[256];
ClObject C_part_notes_path;
      ClAVCreate((&c_part_notes_path, CL_TYPE_STRING,
                  "c_part_notes_path");

      /* Code where ClAVSetStringVal ( ) is used */
      /*                ...                        */

      /* Later */
      c_tp = ClAVGetStringVal(C_part_notes_path);

/* Put transient HliString into something more permanent */
      if(c_tp != (CString) NULL) {
        strcpy(c_temp_parts_notes_path, c_tp);
      }
```

Notes

Since CL_TYPE_STRING can refer to either a HliString or a CString in the create function, ClAVCreate(), it is up to the application developer to keep track of which GetString function to call. Use Naming conventions on variables to avoid confusion.

See Also

Notes on transient strings under ClAVGetHliString ().

ClAVSetStringVal ()
ClAVGetHliStringVal ()
ClAVSetValue ()
ClAVGetValue ()

ClAVGetValue

Purpose

Get the value of an active variable previously defined by `ClAVCreate()`. For anything but the default value, one of the CLAVSet functions must also have been called.

Syntax

```
HliStatus ClAVGetValue
            (ClAVObject av,
            void *value)
```

Description

`av` is an active variable handle returned from `ClAVCreate()`.

`value` is a pointer and can point to one of the following types `CL_TYPE_INTEGER`, `CL_TYPE_STRING`, `CL_TYPE_FLOAT`, or `CL_TYPE_DOUBLE`.

Example

```
/* This example assumes an active variable,"c_part_notes_path" has
already been created to hold a CString and that some file name has
been assigned using
CLAVSetStringVal(). */

FILE *out;
char file_to_open[256];
void *void_ptr_val;
HliStatus err_stat;

      err_stat = ClAVGetValue(c_part_notes_path,
                        void_ptr_val);

      if(HliStatusIsSuccess_(err_stat)){
            strcpy(file_to_open, ((CString) void_ptr_val));
            if((out = fopen(file_to_open, "w")) == NULL) {
                  fprintf(stderr, "Can't open file %s\n",
                        file_to_open);
            }
      }
```

Notes

In this example note the cast in the call to `strcpy()`. Since the value to be assigned to `void_ptr_val` can be one of several types, a pointer to void must be used in the `ClAVGetValue()` call. However, when the value is passed to `strcpy`, or used anywhere a CString is expected, it must be cast into the appropriate type.

ClAVRemoveCallback

Purpose

Undo `ClAVAddCallback()`, i.e., unregister a callback.

Syntax

```
HliStatus ClAVRemoveCallback
            (ClAVObject av,
            ClAVCallback callback,
            void *data)
```

Description

This function takes the same args passed to `ClAVAddCallback()`.

`av` is a handle returned with `ClAVCreate()`.

`callback` is the function to be called by `ClAVSetValue()`.

`data` is some data to be passed to the callback; an example would be a UiObject.

See Also

ClAVCreate ()

ClAVSetDoubleVal

Purpose

Set the value of an active variable of type `CL_TYPE_DOUBLE`.

Syntax

```
HliStatus ClAVSetDoubleVal
            (ClAVObject av,
      double value)
```

Description

`av` is the handle returned from `ClAVCreate()`.

`value` is the value to set.

This function is a wrapper for `ClAVSetValue()` for active variables of type `CL_TYPE_DOUBLE`.

Returned errors can be

- Active variable doesn't exist.
- Active variable is not of type `CL_TYPE_DOUBLE`.

See Also

ClAVSetFloatVal ()
ClAVSetIntegerVal ()
ClAVSetValue ()

ClAVSetFloatVal

Purpose

Set the value of an active variable of type `CL_TYPE_FLOAT`.

Syntax

```
HliStatus ClAVSetFloatVal
            (ClAVObject av,
      double value)
```

Description

`av` is the handle returned from `ClAVCreate()`.

`value` is the value to set.

This function is a wrapper for `ClAVSetValue()` for active variables of type `CL_TYPE_FLOAT`.

Returned errors can be

- Active variable doesn't exist.
- Active variable is not of type `CL_TYPE_FLOAT`.

Notes

The value to be set needs to be a double because K&R C passes floats as doubles.

See Also

ClAVSetDoubleVal ()
CLAVSetIntegerVal ()
ClAVSetValue ()

ClAVSetHliStringVal

Purpose

Set the value of an active variable of type `CL_TYPE_STRING`.

Syntax

```
HliStatus ClAVSetHliStringVal
            (ClAVObject av,
      HliString value)
```

Description

`av` is the handle returned from `ClAVCreate()`.

`value` is the value to set.

This function is a wrapper for `ClAVSetValue()` for active variables of type `CL_TYPE_STRING`. Given an HliString, set the value of an active variable of type `CL_TYPE_STRING`.

Returned errors can be

- Active variable does not exist.
- Active variable is not of type `CL_TYPE_STRING`.

See Also

ClAVSetStringVal ()

ClAVSetIntegerVal

Purpose

Set the value of an active variable of type `CL_TYPE_INTEGER`.

Syntax

```
HliStatus ClAVSetIntegerVal
            (ClAVObject av,
      int value)
```

Description

`av` is the handle returned from `ClAVCreate()`.

`value` is the value to set.

This function is a wrapper for `ClAVSetValue()` for active variables of type `CL_TYPE_INTEGER`.

Returned errors can be

- Active variable doesn't exist.
- Active variable is not of type `CL_TYPE_INTEGER`.

See Also

ClAVSetValue ()
ClAVSetFloatVal ()
ClAVSetDoubleVal ()

ClAVSetStringVal

Purpose

Set the value of an active variable of type `CL_TYPE_STRING`.

Syntax

```
HliStatus ClAVSetStringVal
            (ClAVObject av,
      CString value)
```

Description

`av` is the handle returned from **C**`lAVCreate()`.

`value` is the value to set.

This function is a wrapper for `ClAVSetValue()` for active variables of type `CL_TYPE_STRING`. Given a C string, set the value of an active variable of type `CL_TYPE_STRING`.

Returned errors can be

- Active variable does not exist.
- Active variable is not of type `CL_TYPE_STRING`.

See Also

ClAVSetHliStringVal ()

ClAVSetValue

Purpose

Set a variable previously defined by `ClAVCreate()`. Then call all callbacks registered for that variable.

Syntax

```
HliStatus ClAVSetValue()
            (ClAVObject av,
            void *value)
```

Description

`av` is the handle returned from `ClAVCreate()`.

`value` is an address of an integer, float, or double, or it is a pointer to a C string.

Notes

This routine is designed for remote use, but nothing prevents local use. This routine may be confusing to use. If so, use the more specific convenience functions which follow, for setting integers, floats, doubles and strings.

ClBufferCCommand

Purpose

Store a CADDS command in the CADDS command buffer.

Syntax

```
HliStatus ClBufferCCommand(const CChar *format, ...)
HliStatus ClBufferCommand(HliChar *format, ...)
```

Description

The `format` statement is a `printf` style format statement. However, the only format directives allowed currently are `%S` and `%s`

- `%S` means one of the arguments is an HliString
- `%s` means that one of the arguments is a C String

Example

```
int TestBuffered()
{
    HliString hliformat;
    HliString hliarg;
    HliStatus st;
    char *carg3 = "A C String as the argument,\
                        ClBufferCCommand";
    char *carg4 = "A C String as the argument,\
                        ClBufferCommand";

    HliTestAssertStatus_(ClBufferCCommand,
                        ("A C string as the format,\
                        ClBufferCCommand\n"));
    HliTestAssertStatus_(ClBufferCCommand,
                    ("%s\n",
                     carg3));
    hliarg = HliFromCString("An HliString as the argument,\
                        ClBufferCCommand");
    HliTestAssertStatus_(ClBufferCCommand,
                    ("%s, %S\n",
                     carg3,
                     hliarg));
    HliFree_(hliarg);

    hliformat = HliFromCString("An HliString as the \
                        format to ClBufferCommand\n");
    HliTestAssertStatus_(ClBufferCommand,(hliformat));
    HliFree_(hliformat);
```

```
    hliformat = HliFromCString("%s\n");
    HliTestAssertStatus_(ClBufferCommand,

                  (hliformat,
                   carg4));
    HliFree_(hliformat);

    hliformat = HliFromCString("%S, %s\n");
    hliarg = HliFromCString("An HliString as the argument,\
                              ClBufferCommand");
    HliTestAssertStatus_(ClBufferCommand,
                   (hliformat,
                    hliarg,
                    carg4));
    HliFree_(hliformat);
    HliFree_(hliarg);
    HliTestAssertStatus_(ClBufferCCommand,
                   ("ClBufferCCommand, the next one is \
                  ClBufferMsg, No CADDS connection\n"));
      HliTestAssertStatus_(ClBufferMsg,
                        (CL_INF_INTERFACE_ONLY));
            HliTestAssertStatus_(ClBufferCCommand, ("\n"));

    st = CL_BUFFER_VN_(VN_ACTIVATE);

    return st;
}

void testNewSendCommand()
{

    /* printf("Now testing buffered commands\n"); */
    HliTestAssertStatus_(TestBuffered,());

    /* printf("first throw them away with \
                        ClAbortCommand\n"); */
    HliTestAssertStatus_(ClAbortCommand,());

    /* printf("Now queue them up again\n"); */
    HliTestAssertStatus_(TestBuffered,());


    /* printf("next dump buffered commands out with \
                        ClFlushCommand\n"); */
    HliTestAssertStatus_(ClFlushCommand,());

    /* printf("Now queue them up again\n"); */
    HliTestAssertStatus_(TestBuffered,());

    /* printf("next dump them out implicitly by calling \
                        ClSendCCommand\n"); */
    HliTestAssertStatus_(ClSendCCommand,
                   ("\nCommand to force out the \
```

```
                                buffered stuff"));
}
```

Here is the resulting output. This output comes from running a standalone CR process which is not connected to CADDS. It should also work in CVact when CVact is not connected to CADDS.

Notice in the following text that the newlines were explictly placed in the format statements or in the text. Nothing in the function calls automatically put in newlines. You have complete control over whether you want newlines, spaces, control characters, etc.

```
Send Command to cadds
A C string as the format, ClBufferCCommand
A C String as the argument, ClBufferCCommand
A C String as the argument, ClBufferCCommand, An HliString \ as
the argument, ClBufferCCommand
An HliString as the format to ClBufferCommand
A C String as the argument, ClBufferCommand
An HliString as the argument, ClBufferCommand, A C String \ as the
argument, ClBufferCommand
ClBufferCCommand, the next one is ClBufferMsg, No Cadds \
connection
CL_INF_INTERFACE_ONLY No Cadds Connect
ACTIVATE
Send Command to cadds
A C string as the format, ClBufferCCommand
A C String as the argument, ClBufferCCommand

A C String as the argument, ClBufferCCommand, An HliString \ as
the argument, ClBufferCCommand
An HliString as the format to ClBufferCommand
A C String as the argument, ClBufferCommand
An HliString as the argument, ClBufferCommand, A C String \ as the
argument, ClBufferCommand
ClBufferCCommand, the next one is ClBufferMsg, No Cadds \
connection
CL_INF_INTERFACE_ONLY No Cadds Connect
ACTIVATE
Command to force out the buffered stuff
```

Notes

`ClBufferCCommand`, `ClBufferCommand`, and `ClBufferMsg` concatenate together the command fragments into one long command, which will be sent later. They have the advantage of letting you analyze piece by piece the data that you have to send, without having to manage the memory allocation for the command fragments yourself.

ClBufferMsg

Syntax

```
HliStatus ClBufferMsg(HliStatus msgId)
```

Description

The input to `ClBufferMsg` would be one of the symbol constants defined in a `.h` file generated from a UIHLI `.mrs` file.

Notes

`ClBufferCCommand`, `ClBufferCommand`, and `ClBufferMsg` concatenate together the command fragments into one long command, which will be sent later. They have the advantage of letting you analyze piece by piece the data that you have to send, without having to manage the memory allocation for the command fragments yourself.

ClCaddsRtlInstanceWrap

Purpose

Convenience function for accessing ANY function of type `RTL_instance` in CADDS. (`RTL` stands for RunTime List.)

Syntax

```
HliString ClCaddsRtlInstanceWrap
            (const char *func,
            HliSignedLong index)
```

ClCaddsRtlSizeWrap

Purpose

Convenience function for accessing the size wrap function in CADDS.

Syntax

```
HliSignedLong ClCaddsRtlSizeWrap (const char *func)
```

Cl_call_CR_func

Purpose

Used for directly calling a CR function from CADDS. This communication interface enables you to design interactive menus like New NURBS and Sketcher of the CADDS environment for highly interactive applications.

Now you can launch a CVact menu from CADDS algorithms and also pass the field contents using a single cell.

Syntax

```
HliStatus Cl_call_CR
            (int block,
            ClType retType,
            void *retVal,
            char *rtn,
            ...
            )
```

Description

The call to CR function blocks CADDS, that is, CADDS is blocked until the CR process is complete. But CADDS may also send a request to CR function to complete the process and may not wait for a reply from CR.

Functions in CR callable using the CADDS-CR communication interface is registered by a dynamic registration mechanism. The CADDS-CR communication interface is identical to the ClRequest interface. Use ClRequest to initialize the menu and retrieve information on the menu field contents.

The description of the function is as follows:

- Return data types : `void, integer, float, double, string`
- Argument data types : `integer, float, double string`
- Function argument passing can follow any of the following conventions : pass by value (`CL_MODE_IN`), pass by reference with only return value expected (`CL_MODE_OUT`) and pass by reference with the intention of passing a value and expecting a modified return value (`CL_MODE_INOUT`).

The input for `Cl_call_CR_func` is as follows:

Table C-1

int block	Block for call completion flag
ClType retType	Return value type of CR function
void *retVal	Return value pointer
char *rtn	Name of Cr function to be called
va_alist	Variable argument list. Corresponds to arguments to be passed to the CR function. Maximum of MAX_CR_FUNC_ARGS arguments can be passed. Arguments to be passed are triplets of the form : ClType, ClMode and Value

The output of `Cl_call_CR_func` is as follows:

```
void *retVal Return value of CR function
```

The return is as follows:

```
OUR_SUCCESS / OUR_FAILURE
```

Examples

The examples given explain the usage of the functionality.

```
st = cl_call_CR_func
(TRUE, CL-type-INTEGER, &ret, “dispPtonMenu”,
CL_TYPE_INTEGER, CL_MODE_IN, id,
CL_TYPE_INTEGER, CL_MODE_IN, tang_display,
CL_TYPE_DOUBLE, CL_MODE_IN, Z,
CL_TYPE_VOID
```

The example for the function being called is as follows:

```
int DispPtonMenu
(int argCount, FUNC_ARGS args [])
{
UiObject obj, parobj;
HliStatus st;
/* Set up the Pton menu id*/
ptonMenuId = args [0].argVal.i;

/* Launch the Pton menu */
st = UiGetObjectByName (“PtonCurveHandle”,UI_NULL_OBJECT,&parobj);
if (st != HLI_OK)
parobj = create_PtonCurveHandle (UI_NULL_OBJECT);
st = UiGetObjectByName (“Toggle1”,parobj, &obj);
if (st == HLI_OK)
UiSetDefaultBoolean (obj, (HLIBoolean) args [1] .argVal . i);
st = UiGetObjectByName (“Toggle2”,parobj,&obj);
if (st == HLI_OK)
```

```
UiSetDefaultFloatValue (obj,(HLIBoolean) args [2] .argVal . d);

st = ClLaunchDialogInPlace ("PtonCurveHandle");
if (st != HLI_OK)
return (-1);
else
if (st == HLI_OK)
return (0);
}
```

Notes

Other CADDS-CR communication functions are as follows:

- For return data type `void` :

```
cl _av_setvalue (char *name, ClType, void *value)
Call suim to set active variable in CR
```

- For return data type `string`:

```
cl_av_set_string_value (char *name, char *value)
Call suim to set CL_TYPE_STRING active variable in CR
```

- For return data type `integer`:

```
cl_av_set_integer_value (char *name, int value)
Call suim to set CL_TYPE_INTEGER active variable in CR
```

- For return data type `float`:

```
cl_av_set_float_value (char *name, float value)
Call suin to set CL_TYPE_FLOAT active variable in CR
```

- For return data type `double`:

```
cl_av_set_double_value (char *name, double value)
Call suim to set CL_TYPE_DOUBLE active variable in CR
```

- For launching the menus:

```
cl_launch_dialog (char * menu)

cl_launch_dialog_in_place (char *menu)
```

ClClearPrompt

Purpose

Used for Parametrics or CVdesign to clear the prompt line.

Syntax

```
HliStatus ClClearPrompt()
```

Cl_E_epsilon_value

Purpose

Wrapper function for fetching `E_epsilon_value`. Tolerancing value for doing Booleans or some such.

Syntax

```
HliDouble Cl_E_epsilon_value()
```

Cl_E_extents_value

Purpose

Wrapper function for fetching E_extents_value. Extents of drawing.

Syntax

```
HliDouble Cl_E_extents_value ()
```

ClFlushCommand

Purpose

`ClFlushCommand` sends to CADDS all of the buffered command information that you may have stored with ClBufferCCommand, `ClBufferCommand` and `ClBufferMsg`.

Syntax

```
HliStatus ClFlushCommand()
```

Notes

`ClSendCCommand` and `ClSendComand` will also have the affect of sending the previously buffered information to CADDS.

ClGetDialogInstances

Purpose

Gets all instances for the specified dialog box or property sheet.

Syntax

```
HliStatus ClGetDialogInstances
(const char *dialogName,
UiObject *instances);
```

Description

The Cl function `ClRegisterDialogMaxInstances` allows the creation of multiple instances of the specified property sheet or dialog box. The maximum number of instances allowed for a property sheet or dialog box is in the range 1 to 65,535.

Use `ClGetDialogInstances` to get all the instances created for the specified property sheet or dialog box.

ClGetDialogLastInstanceId

Purpose

Gets the last instance id created for the specified dialog box or property sheet.

Syntax

```
HliStatus ClGetDialogLastInstanceId
(const char *dialogName,
HliUnSignedShort *last_instance_id);
```

Description

The Cl function `ClRegisterDialogMaxInstances` allows the creation of multiple instances of the specified property sheet or dialog box. The maximum number of instances allowed for a property sheet or dialog box is in the range 1 to 65,535.

The instance id is in the range 1 to 65,535. The Cl Library automatically sets the appropriate instance id on each of the multiple instances of the dialog box or property sheet, launched either from the CADDS algorithms using `cl_launch_dialog` or from the CR side using the `ClLaunchDialog` binding.

Use `ClGetDialogLastInstanceId` to get the last instance id created for the specified property sheet or dialog box.

See Also

- ClGetDialogNumCurInstances
- ClGetDialogMaxInstances

ClGetDialogMaxInstances

Purpose

Gets the maximum instances that can be created for a specified dialog box or property sheet. Specify the dialog box or property sheet by name.

Syntax

```
HliStatus ClGetDialogMaxInstances
            ((const char *dialogName,
            HliUnsignedShort *max_instances));
```

Description

`ClRegisterDialogMaxInstances` allows the creation of multiple instances of a CVact menu.

Gets the maximum instances that can be created for the specified property sheet or dialog box. Specify the dialog box or property sheet by name.

See Also

- ClRegisterDialogMaxInstances
- ClGetDialogLastInstanceId
- ClGetDialogNumCurInstances

ClGetDialogNumCurInstances

Purpose

Gets the current number of instances existing for the specified dialog box or property sheet.

Syntax

```
HliStatus ClGetDialogNumCurInstances
            (const char *dialogName,
            HliUnsignedShort *cur_instances);
```

Description

`ClRegisterDialogMaxInstances` allows the creation of multiple instances of a CVact menu.

You can get the current number of instances existing of a property sheet or dialog box with `ClGetDialogNumCurInstances`. Specify the dialog box or property sheet by name.

`ClGetDialogNumCurInstances` reports the actual number of instances existing for a property sheet or dialog box. The number of existing instances reported by `ClGetDialogNumCurInstances` may be different from the last instance id reported by `ClGetDialogLastInstanceId`.

See Also

- ClGetDialogLastInstanceId
- ClGetDialogMaxInstances

ClGetFloatStringVal

Purpose

Fetch CADDS-style (U.S.) text representation of a float field.

Syntax

```
HliString ClGetFloatStringVal(UiObject obj)
```

Notes

CADDS can only handle U.S. format integers and floats.

ClGetIntegerStringVal

Purpose

Fetch CADDS-style (U.S.) text representation of an Integer field.

Syntax

```
HliString ClGetIntegerStringVal(UiObject obj)
```

Notes

CADDS can only handle U.S. format integers and floats.

ClInitialize

Purpose

This initializes Cl, and should be called before any other Cl routines.

Syntax

```
HliStatus ClInitialize
            (int *argc,
            char *argv[])
```

Example

This function removes any arguments it cares about from `argv`. If we aren't really running CADDS (just testing) then it sets ClInterfaceOnly and returns `CL_INF_INTERFACE_ONLY`.

ClLaunchDialog

Purpose

Launch a dialog that has been previously registered with `ClRegisterDialog()`. This routine is designed for remote use (from Composer). But nothing prevents local use.

Syntax

```
HliStatus ClLaunchDialog ((const char *name))
```

ClRegisterDialog

Purpose

Make a dialog launchable from `ClLaunchDialog()`, and therefore by Composer.

Syntax

```
HliStatus ClRegisterDialog
            (const char *dialogName,
            UiObject handle,
            ClBuildDialogRtn buildRtn,
            void *buildRtnData)
```

Description

You can pass the dialog in directly, with the `handle` arg. Or you can have "lazy creation" where the dialog is not created until the first time it is launched—to do this, pass `buildRtn`, which will be called with `buildRtnData` as its only argument.

The `handle` is NULL, or the handle returned from calling the build function first at initialization time. In most situations, it will be NULL.

The `buildRtn` is NULL or the function to call when `ClLaunch()` is called. If handle == NULL, then `buildRtn` must be something. If handle != NULL, then `buildRtn` should be NULL.

The `buildRtnData` is NULL, or a pointer to a structure or array that is not on the stack.

ClRegisterDialogMaxInstances

Purpose

Allows multiple instances of a specified dialog box or property sheet. Specify the dialog box by name. If `ClRegisterDialogMaxInstances` is not called, by default, the creation of only a single instance of the dialog box or property sheet is allowed.

Syntax

```
HliStatus ClRegisterDialogMaxInstances
            (const char *dialogName,
            HliUnsignedShort max_instances);
```

Description

`ClRegisterDialogMaxInstances` allows the creation of multiple instances of a CVact menu. Specify the dialog box or property sheet by name.

You can create as many as 65,535 instances of the same property sheet or dialog box. You can also have specific settings on each of these multiple instances.

Tack each instance of the dialog box or property sheet to retain it on screen and launch the menu again to create a new instance. If all the existing instances of the property sheet or dialog box are visible and tacked, then a new instance of the property sheet or dialog box is created each time you try to launch the property sheet or dialog box.

If the maximum number of instances of the property sheet or dialog box already exists, and are visible and tacked, a new instance of the property sheet or dialog box is not created. Instead, the following error message is displayed in the command window:

`Max instances for dialog` DialogName `exist! Can not create a new instance`

If you attempt to launch the property sheet or dialog box when any one of the existing instances is not tacked and is not visible, then a new instance of the property sheet or dialog box is not created. Instead, an instance of the dialog box which is already existing but is not tacked and is not visible is made visible. You may reset settings on this instance and retain it by tacking it.

Each of the multiple instances is automatically assigned an instance id when the property sheet or dialog box is launched from the CADDS algorithms using `cl_launch_dialog`, or is launched from the CR side using `ClLaunchDialog`. The instance id is also in the range 1 to 65,535. In a session, the instance id, once assigned, remains the same and does not change.

Use `UiGetInstanceId` to get the instance id for a specific instance of the property sheet or dialog box.

Example

In Assembly mode, the View Component property sheet which is a CVact menu, can have multiple instances limited to a maximum of 65, 535 instances.

The example given shows the following use of the functionality:

```
/*Set max instances for View/Unview component
   menu to HLI_USHORT_MAX (65535).
*/
ClRegisterDialogMaxInstances("CamuViewUnviewComponentCab",HLI_USH
ORT_MAX);
```

See Also

- ClGetDialogMaxInstances
- UiGetInstanceId

ClRequest

Purpose

Call a CADDS C subroutine.

Syntax

```
HliStatus ClRequest
            (HliBoolean block,
            ClType retType,
            void *retVal,
            const char *rtn,
            ...
            )
```

Description

`Varargs` are up to CL_MAX_ARGS

- triplets ClType, ClMode, value
- quadruplets `CL_TYPE_STRINGBUF`, `int (buf_size)`, `ClMode`, `char *stringbuf`
- quintuplets `CL_TYPE_ARRAY`, `ClType (elt type)`, `ClMode`, `int (nelements)`, `void * (elts)`

followed by `CL_TYPE_VOID`.

Cl_RTL_caddspaths_size

Purpose

Wrapper function for fetching the path size of a CADDS RTL.

Syntax

```
HliSignedLong Cl_RTL_caddspaths_size ()
```

Cl_RTL_e_drawing_name_init

Purpose

Reinitializes list of Explicit drawing names.

Syntax

```
HliSignedLong Cl_RTL_e_drawing_name_init()
```

Example

```
Cl_RTL_e_drawing_name_init();
```

Notes

This makes sure that CADDS has all of the latest drawings available to be fetched.

Cl_RTL_e_drawing_name_instance

Purpose

Fetches a particular drawing name from CADDS.

Syntax

```
HliSignedLong Cl_RTL_e_drawing_name_instance ()
```

Example

```
UiObject explicitDrawings;
HliSignedShort j, n;

/* Fetch every drawing and add it to the scroll list */
for(j=0; j < n; j++)
      UiAddItem(explicitDrawings,
                  j, Cl_RTL_e_drawing_name_instance(j));
```

Cl_RTL_e_drawing_name_size

Purpose

Returns the number of drawings in CADDS runtime list.

Syntax

```
HliSignedLong Cl_RTL_e_drawing_name_size ()
```

Example

```
HliSignedShort n;
n = (HliSignedShort) Cl_RTL_e_drawing_name_size();
```

Notes

Useful for finding the number of drawing names so that you can put them into a scroll list. See `Cl_RTL_e_drawing_name_instance()`.

Cl_RTL_e_nfig_all_size

Purpose

Wrapper function for fetching the size of all nfigs in a CADDS RTL. Nodal figure.,

Syntax

```
HliSignedLong Cl_RTL_e_nfig_all_size()
```

Cl_RTL_e_tag_all_size

Purpose

Wrapper function for fetching size of all tags in a CADDS RTL.

Syntax

```
HliSignedLong Cl_RTL_e_tag_all_name_size((void))
```

Cl_RTL_e_view_name_size

Purpose

Wrapper function for fetching the view-name size of a CADDS RTL.

Syntax

```
HliSignedLong Cl_RTL_e_view_name_size((void))
```

Cl_RTL_caddspaths_get_name

Purpose

Convenience function for accessing the `caddspaths` suffix part of the CVPATH string. In effect, the function chops off the first six characters of any HliString.

Syntax

```
HliString Cl_RTL_caddspaths_get_name
            (const HliString name)
```

Description

`name` is the literal CVPATH string which is prefixed by "CVPATH."

`return value` is a transient HliString consisting of the string value of the path without the CVPATH prefix.

Notes

The number is hard coded, on the CADDS side, and is manually changed if the number of characters in the environment variable is changed.

ClSendCCommand

Purpose

Send a command to CADDS.

Syntax

```
HliStatus ClSendCommand
            (const HliChar *format,
            ...)

HliStatus ClSendCCommand
            (const CChar *format,
            ...)
```

Description

This will be directed to either the explicit or parametric command lines, as appropriate. `Format` is a format statement with optional arguments, as in HliStringFormat.

The format statement in some of these is a `printf` style format statement. HOWEVER, the only format directives allowed currently are %S and %s. %S means one of the arguments is an HliString. %s means that one of the arguments is a C String.

Notes

`ClSendCCommand` allows you to just put together format statements, such as `%S %S %s`. You are likely to use this function if you have a whole bunch of HliString variables containing the pieces of your command to CADDS.

`ClSendCommand` requires that the format string be an HliString. This would be useful if instead you used the function `HliStrcat` to put together the components of your command, before calling `ClSendCommand`. In this case, you just have an HliChar *format, and wouldn't have any arguments.

ClSendErrorCode

Purpose

Pass an error code to CADDS.

Syntax

```
HliStatus ClSendErrorCode
            (HliStatus status,
            ...)
```

ClSendErrorCString

Purpose

Pass an EUC error string to CADDS.

Syntax

```
HliStatus ClSendErrorCString(const char *error)
```

ClSendErrorHliString

Purpose

Pass an HLI error string to CADDS.

Syntax

```
HliStatus ClSendErrorHliString(const HliChar *error)
```

ClSendWarning

Purpose

Pass an HLI error/warning message string to CADDS.

Syntax

```
HliStatus ClSendWarning(const HliChar *message)
```

ClSetPrompt

Purpose

Used for CADDS to set the prompt line.

Syntax

```
HliStatus ClSetPrompt(const HliChar *prompt)
```

Example

```
HliString prompt;
      prompt = HliFromCString("This is a user prompt");
      ClSetPrompt(prompt);
      HliFree_(prompt);
```

Glossary

Action

An action is a named section of C code (used with Translation Tables). You define the name of the action and the associated C code by means of the Action Table. Then, using the Translation Table Editor, you specify the actions which are to be executed when specified events occur.

Action Table

The Action Table allows you to define action names and assign corresponding code. Actions can then be linked to an event in the Translation Table.

Adjust Button

On a three-button mouse, the Adjust button is the middle one. On a two-button mouse, pressing both buttons at once is equivalent to pressing the Adjust button.

Application

An application is a program developed by you. CVact is used to add an OSF/Motif-based user interface to your application.

Application Defaults

In CVact, a number of resources can be set to specify user preferences. These resources can be used to define a different look and feel for CVact and to also set defaults. Application Defaults can be set system-wide (for all users of the system), for a single user, or for a single instance of CVact.

Callback

A callback is a piece of C code associated with a widget or gadget which is executed when a specified event occurs. For example, a pushbutton has an activate callback, which is executed when the button is pressed and released. A toggle button has two callbacks, one for when it is selected, and one for when it is deselected.

Compass Pointer

The special mouse pointer, or cursor, that appears when you press and hold down the Adjust (middle) mouse button.

Component

A component is a top-level interface that you reuse in other interfaces. In object-oriented terminology, a Component is a class.

Constraint set

A Constraint set is a PList of the resource descriptors for the constraint properties defined by a constraining manager widgets. Each child of a constraining manager has its own Constraint set.

Developer

The developer is the person who uses CVact to produce the final product or interface that lets the user interact with an application.

Drag and Drop

Drag and drop is the process that enables you to select one object on your screen with the mouse, and then move it on top of another object to perform some function. For instance, in CVact you can drag and drop any widget into the Property Editor to load its properties. Using drag and drop makes working faster and more visual.

Event

An event occurs when the user communicates with the program. Typical events are button presses, button releases, keyboard key presses, and mouse movement. CVact allows you to interactively prototype the behavior of an interface by defining what happens when specified events occur.

Expressions list

An Expressions list is a PList of initial value expressions. An initial value expression yields the initial value of a property (the value given to the property when the swidget is first created). Each swidget has its own expressions list.

Icon

An icon is a small graphic used as a visual representation of an action you can perform or an object you can manipulate. For instance, in CVact an icon appears in the Project Window for each of your interfaces.

Insensitive

An insensitive widget or selection does not respond to user input.

Instance

An Instance is a use of a Component. In object-oriented terminology, an Instance is an object of a class.

Interface

An interface is a top-level widget together with all of its descendants. Normally, an interface is used to control a particular aspect of an application. For example, an interface may appear when the user asks the application to read a file. This interface would contain a text widget for typing in the file name, a pushbutton widget to initiate the read operation, and label widgets to identify the function of the interface.

Interface File

An interface file stores all aspects of one or more interfaces in an X-like resource file format consisting of a header and a sequence of X-style resource specifications. The resource specifications in an interface file set the properties of one or more objects. CVact can generate an interface file and then read it back in at a later date.

Interface files serve as a permanent archive for storing interfaces. CVact generates an interface file when you save an interface, a project, or a palette. .i interface files define interfaces, .prj interface files define projects, and .pal interface files define palettes.

It is important to remember that C code generated by CVact can only be read in and executed, not edited, so you need the interface file to make any further changes to the interface. Interface files normally have an extension of .i or .act.

Interface Function

The C file which CVact generates for an interface contains an interface function which is called to create (and optionally popup) the interface. The Declarations Editor allows you to specify parameters to the interface function and to add extra code to be executed before or after the interface is created.

Interpreter

CVact contains a C Interpreter. By means of the Interpreter, C code which you type in can be executed immediately without recompilation. In addition, if your application is linked with CVact, then all of the application's functions are also available through the Interpreter. In this way, prototyping the behavior of the user interface is rapid, since no recompilation or relinking is necessary.

Menu bar

A horizontal bar that contains a list of menus. The menu bar is displayed below the window title area.

Menu Button

On a two- or three-button mouse, the Menu button is the right one.

Modal dialog

A dialog that requires a response from the user before work can continue.

Mouse Pointer

The cursor, or arrow on the screen that shows your active location.

Observer

An editor or tool that receives an update message when certain actions are performed on a swidget.

Palette

A tool or area containing one or more widgets or widgets groups that you can quickly apply to your interface. A default Palette is provided with CVact, or you can easily create your own custom Palettes.

The widget palette that appears by default at startup is an example of a palette.

Project

A project is a set of interfaces designed for a single application. Project files normally have an extension of .prj.

Project Window

The Project Window is the fundamental window that appears when you start CVact. Using the Project Window enables you to create, store, load, and test your interfaces.

Property

In CVact, a property is a widget variable. Properties define the appearance and behavior of widgets. The set of widget properties defined by CVact is a superset of the widget resources defined by Motif. CVact defines additional widget properties such as the compound properties to help manage and support GUI development.

Some examples of properties are width, height, color, and the activate callback function. You can change these values using CVact's Property Editor. Properties is the CVact name for what are more commonly known as resources. The term properties used by CVact is not to be confused with the X Window System's use of that term.

Radio Buttons

A radio button consists of a button and a label that specifies the choice associated with the button. Radio buttons are combined to provide a fixed set of mutually exclusive choices.

Recreate

To destroy (unrealize) and create (realize).

Resource Descriptor

A data structure associated with a class property.

Resource Set

A PList of pointers to the resource descriptors defined for the properties of a swidget class.

Select Button

On a two- or three-button mouse, the Select button is the left one.

Selection list

A PList of the swidgets currently selected by the user.

Shell

Shell widgets correspond to the X Toolkit shell widgets. Shells are automatically created by CVact when a top-level widget is created, but can also be explicitly created.

Submenu

A menu displayed when a cascade button is pressed.

Top-level Widget

At the root of each widget hierarchy that you create is a top-level widget which interacts with the window manager. Top-level widgets contain children. For example, a top-level widget such as an application shell can contain a bulletin board widget, which in turn can hold a number of push buttons. In CVact, manager widgets can be top-level widgets because CVact can automatically create a (hidden) shell widget.

Translation Table

The Translation Table contains an event specification and the associated action.

User

The user is the person who uses the final product produced by the developer.

User swidget

A swidget created interactively in CVact or loaded into CVact as part of an interface file.

Widget

A widget is a basic component of a user interface. Each widget has a set of built-in properties and behaviors which make it suitable for some aspect of the interface. Thus, there are pushbutton widgets, toggle widgets, bulletin board widgets, and so on. CVact provides you with access to the complete set of OSF/Motif Widgets.

Index

Symbols

(Label)
  Icon(cust) 5-23
.act file 4-2
.caddsrc file 4-4
.CR file 4-15

A

AbortCb callback 5-2
  aborting active tool operation 5-2
  ToolEntry 5-2
Accelerator(cust) attribute 5-3
  CommandEntry 5-3
  DirectActionEntry 5-3
  SpecifierEntry 5-3
  specifying keyboard accelerator 5-3
  ToggleEntry 5-3
  ToolEntry 5-3
ActionEntry class 6-8
ActivateCb callback 5-3
  activating entry 5-3
  Entry 5-3
Active variables
  CADDS process C-7
  CL bindings
    from CADDS C-8
    from CR C-9
  CR process C-7
  from CR C-9
  types C-8
  using between CADDS, CVDORS and CR process C-7
ActiveWindow attribute 5-3
  current active window 5-3
  UiObject 5-3
Adding objects
  to UIObjects
    palette 2-3
Advanced Customization Tool 1-2
  customizing
    interface 1-2
  window 1-7
AlignCmdBox
  ApplyLabel(cust) 5-5
  WindowTitle(cust) 5-43
Aligning Form
  Gutter 5-21
AligningForm
  ConstantHeight 5-12
  ConstantWidth 5-13
  NumColumns(cust) 5-31
  Title(cust) 5-41
AligningForm class 6-9
  used as container of fields 6-9
Alignment attribute 5-3
  Label 5-3
  Push button 5-3
  specifying where to align text 5-3
AlternateIcon(cust) attribute 5-4
  Push button 5-4
  showing button is toggled 5-4
  Toggle 5-4
appinitU.cxx file 4-4
Application code 1-5
ApplicationDesktop
  AssistOn(cust) 5-6

CommandOn(cust) 5-11
CurrentTool(cust) 5-13
Name 5-30
Parent 5-32
Prompt 5-33
PromptOn(cust) 5-33
ApplicationDesktop class
parent for all dataset windows 6-11
ApplicationState
Name 5-30
State 5-39
ApplyButton attribute 5-4
CommandBox 5-4
used so that you can keep non-modal boxes posted while doing other things 5-4
ApplyCb callback 5-4
called when the OK or Apply button is clicked 5-4
CommandBox 5-4
ApplyLabel(cust) attribute 5-5
AlignCmdBox 5-5
CommandBox 5-5
FromMessage convenience functions 5-5
text used to label the Apply button 5-5
ApplyPulldown class 6-12
provides an extra level of interaction for the user 6-12
ArgumentBox attribute 5-5
CommandEntry 5-5
used to look up the corresponding command argument box by name 5-5
AssistLIne(cust) attribute
UiObject 5-5
AssistLine(cust) attribute 5-5
FromMessage convenience functions 5-5
text shown in the assistance area when mouse or cursor lies on top of this object 5-5
AssistOn(cust) attribute 5-6
ApplicationDesktop 5-6
indicates whether assist area is to be made visible 5-6
Attribute
Accelerator(cust) 5-3
ActiveWindow 5-3
Alignment 5-3
AlternateIcon(cust) 5-4
ApplyButton 5-4
ApplyLabel(cust) 5-5
ArgumentBox 5-5
AssistLine(cust) 5-5
AssistOn(cust) 5-6
Background 5-6
ButtonParent 5-6
CancelLabel 5-7
Cell 5-7
CellValues 5-7
Checked(cust) 5-7
Chosen 5-8
ChosenItems 5-9
ChosenItemsNum 5-9
ChosenObject 5-9
Column Alignments 5-10
ColumnLabel 5-10
ColumnLabels 5-10
ColumnsNum 5-10
ColumnsWide 5-11
ColumnWidths 5-10
CommandOn(cust) 5-11
ConstantHeight 5-12
ConstantWidth 5-13
Contents(cust) 5-13
ContentsNum 5-13
CurrentTool(cust) 5-13
CursorPosition 5-14
DeactivateCb 5-14
DecimalPlaces(cust) 5-14
DefaultBoolean(cust) 5-15
DefaultButton 5-15
DefaultFloatValue(cust) 5-15
DefaultIntegerValue(cust) 5-15
DefaultItems 5-16
DefaultItemsNum 5-16
DefaultObject 5-16
DefaultString(cust) 5-16
DefaultStringNum 5-17
Defective 5-17
Dependency 5-17
DependencyExpression 5-18
DependencyType 5-18
Editor 5-19
EditorAttributes 5-19
Entry 5-20
FloatValue 5-20
FontCategory 5-21
Foreground 5-21
Framed 5-21
Gutter 5-21
Height 5-22
HeightSpecified 5-22
HelpContext 5-22
HLS(cust) 5-23

Inline(cust) 5-24
InputPermitted 5-24
IntegerValue 5-25
Item 5-26
ItemsNum 5-26
LabelFontCategory 5-26
list 5-2
Manage 5-26
MaxFloatValue(cust) 5-27
MaxIntegerValue(cust) 5-27
MaxLength 5-27
MaxVisibleLength 5-27
MaxVisibleLines 5-28
MinFloatValue(cust) 5-28
MinIntegerValue(cust) 5-28
MinVisibleColumns 5-28
MinVisibleLength 5-29
MinVisibleLines 5-29
MinVisibleRows 5-29
Modal 5-29
NumColumns(cust) 5-31
OkLabel(cust) 5-31
Orientation 5-31
OtherData 5-31
partial syntax 5-2
PercentDrawn 5-32
Pointer 5-32
PositionInDesktop 5-32
PromptOn(cust) 5-33
RadioBehavior 5-34
Remembered 5-34
RememberedBoolean 5-34
RememberedItems 5-35
RememberedItemsNum 5-35
RememberedObject 5-35
RememberedString 5-35
RememberedStringNum 5-35
ResetLabel 5-36
RGB 5-34
RowColumn 5-36
RowLabel 5-37
RowLabels 5-37
RowsHigh 5-37
RowsNum 5-37
Sensitive(cust) 5-37
SensitiveNow 5-38
Source(cust) 5-38
Specifier(cust) 5-38
StartColumn 5-39
StartRow 5-39
State 5-39
StatusArea 5-39
Sticky 5-39
TextFontCategory 5-40
TextString 5-40
TextStringNum 5-40
TextValue(cust) 5-40
Title(cust) 5-41
UserText(cust) 5-41
VisibleNow 5-42
VisibleWhenManaged(cust) 5-42
Width 5-42
WidthSpecified 5-43
WindowTitle(cust) 5-43
XPosition 5-44
XSpecified 5-44
YPosition 5-44
YSpecified 5-45
Attributes
customizing 3-30

B

Background attribute 5-6
background color category 5-6
BuildingBlock 5-6
Bit mask
MouseEventMask 5-30
BuildingBlock
Background 5-6
Foreground 5-21
Height 5-22
HeightSpecified 5-22
HelpContext 5-22
Manage 5-26
Parent 5-32
PositionInDesktop 5-32
RGB color type 5-34
RowColumn 5-36
Sensitive(cust) 5-37
SensitiveNow 5-38
Visible 5-41
VisibleNow 5-42
VisibleWhenManaged(cust) 5-42
Width 5-42
WidthSpecified 5-43

XPosition 5-44
XSpecified 5-44
YPosition 5-44
YSpecified 5-45
ButtonParent
object into which application-created push buttons should be parented 5-6
ButtonParent attribute 5-6
CommandBox 5-6

C

CADDS
sending commands C-4
CADDS 5 desktop
iconization
CI_C5_iconized variable 4-17
CADDS applications
customizing 1-5
CADDS functions
public interfaces C-6
CADDS process
active variables C-7
CADDS_LAF 4-4
cadds5 command 4-4
Calculator
FloatFillin 1-17
IntegerFillin 1-17
inverse functions 6-32
Callback
AbortCb 5-2
ActivateCb 5-3
ApplyCb 5-4
CancelCb 5-6
ChosenCb 5-8
CompleteCb 5-11
CompleteCellCb 5-12
DefaultActionCb 5-14
DestroyCb 5-18
DropCb 5-18
EditorFocusChangeCb 5-19
EditorInvokedCb 5-20
EditorRescindCb 5-20
IntitializeCb 5-24
IsEditableCb 5-25
IsInputOkCb 5-25
MouseEventCb 5-30
PropertiesCb 5-33
PushedCb 5-33
ResetCb 5-36
WindowActivateCb 5-43
Callback ClosedCb 5-9
Callbacks 1-5, 3-17
clientData 3-22
creating
simple Dialog Box 3-24
HLIString Manipulation utilities 3-22
passing arguments 3-20
systemData 3-22
thisObj 3-22
types 3-17
using 3-20
using the UI Callback Editor 3-23
CancelCb callback 5-6
called when the Cancel button is clicked 5-6
CommandBox 5-6
CancelLabel attribute 5-7
CommandBox 5-7
FromMessage convenience functions 5-7
text used on the Cancel button of a Command Box 5-7
Cell attribute 5-7
ScrollMatrix 5-7
text that can set for an individual cell 5-7
CellValues attribute 5-7
two dimension array of values in the cells of the matrix 5-7
Changing
interface 3-1
layout 3-1
Checked(cust) attribute 5-7
checks toggle, if TRUE 5-7
Toggle 5-7
Children instance 5-8
container 5-8
ChildrenNum item 5-8
Container 5-8
Chosen attribute 5-8
indicates whether an element in a scroll list is chosen 5-8
ScrollList 5-8
ChosenCb callback 5-8
issued when the state is changed 5-8
OptionList 5-8
RadioSet 5-8
ScrollList 5-8
Toggle 5-8
ChosenItems attribute 5-9

array of zero-based position numbers showing which items are chosen 5-9
ScrollList 5-9
ChosenItemsNum attribute 5-9
number of values in ChosenItems array 5-9
ScrollList 5-9
ChosenObject attribute 5-9
object which is the current choice 5-9
RadioSet 5-9
CI bindings
active variables for CADDS C-8
CI function
CI_call_CR_func C-38
CI_E_epsilon_value C-42
CI_E_extents_value C-43
CI_RTL_caddspaths_get_name C-64
CI_RTL_caddspaths_size C-57
CI_RTL_e_drawing_name_init C-58
CI_RTL_e_drawing_name_instance C-59
CI_RTL_e_drawing_name_size C-60
CI_RTL_e_nfig_all_size C-61
CI_RTL_e_tag_all_size C-62
CI_RTL_e_view_name_size C-63
CIAbortCommand C-10
CIAVAddCallback C-12
CIAVCreate C-14
CIAVGetDoubleVal C-16
CIAVGetFloatVal C-17
CIAVGetHliStringVal C-18
CIAVGetIntegerVal C-20
CIAVGetStringVal C-21
CIAVGetValue C-23
CIAVRemoveCallback C-25
CIAVSetDoubleVal C-26
CIAVSetFloatVal C-27
CIAVSetHliStringVal C-28
CIAVSetIntegerVal C-29
CIAVSetStringVal C-30
CIAVSetValue C-31
CIBufferCCommand C-32
CIBufferMsg C-35
CICaddsRtlInstanceWrap C-36
CICaddsRtlSizeWrap C-37
CIClearPrompt C-41
CIFlushCommand C-44
CIGetDialogInstances C-45
CIGetDialogLastInstanceId C-46
CIGetDialogMaxInstances C-47
CIGetDialogNumCurInstances C-48
CIGetFloatStringVal C-49
CIGetIntegerStringVal C-50
CIInitialize C-51
CILaunchDialog C-52
CIRegisterDialog C-53
CIRegisterDialogMaxInstances C-54
CIRequest C-56
CISendCCommand C-65
CISendErrorCode C-66
CISendErrorCString C-67
CISendErrorHliString C-68
CISendWarning C-69
CISetPrompt C-70
communicating with CADDS C-1
notes C-4
CI_call_CR_func CI function C-38
CI_E_epsilon_value CI function C-42
CI_E_extents_value CI function C-43
CI_RTL_caddspaths_get_name CI function C-64
CI_RTL_caddspaths_size CI function C-57
CI_RTL_e_drawing_name_init CI function C-58
CI_RTL_e_drawing_name_instance CI function C-59
CI_RTL_e_drawing_name_size CI function C-60
CI_RTL_e_nfig_all_size CI function C-61
CI_RTL_e_tag_all_size CI function C-62
CI_RTL_e_view_name_size CI function C-63
CIAbortCommand CI function C-10
Class
AligningForm 6-9
ApplicationDesktop 6-11
ApplyPulldown 6-12
CommandBox (CAB) 6-14
Container 6-17
DataSetWindow 6-18
DialogBox 6-19
DirectActionEntry 6-20
EditorDialogBox 6-21
Entry 6-22
EntryList 6-23
ErrorBox 6-24
FailureBox 6-25
Fillin 6-26
FloatFillin 6-28
IdentificationBox 6-30
InformationBox 6-31
IntegerFillin 6-32
Label 6-34
NonAligningForm 6-35

OptionList 6-36
Palette 6-37
Paragraph 6-38
PushButton 6-40
QueryResultBox 6-42
QuestionBox 6-43
RadioSet 6-44
ScrollList 6-45
ScrollMatrix 6-47
Separator 6-49
SlideOffEditorDlgBox 6-50
Slider 6-52
SpecifierEntry 6-53
Toggle 6-54
ToggleEntry 6-55
TopLevel 6-56
WarningBox 6-57
WorkingBox 6-58
Class function
ClAssistOn C-11
Class variable
Definer 5-17
ClAssistOn Cl function C-11
ClAVAAddCallback Cl function C-12
ClAVCreate Cl function C-14
ClAVGetDoubleVal Cl function C-16
ClAVGetFloatVal Cl function C-17
ClAVGetHliStringVal Cl function C-18
ClAVGetIntegerVal Cl function C-20
ClAVGetStringVal Cl function C-21
ClAVGetValue Cl function C-23
ClAVRemoveCallback Cl function C-25
ClAVSetDoubleVal Cl function C-26
ClAVSetFloatVal Cl function C-27
ClAVSetHliStringVal Cl function C-28
ClAVSetIntegerVal Cl function C-29
ClAVSetStringVal Cl function C-30
ClAVSetValue Cl function C-31
ClBufferCCommand Cl function C-32
ClBufferMsg Cl function C-35
ClCaddsRtlInstanceWrap Cl function C-36
ClCaddsRtlSizeWrap Cl function C-37
ClClearPrompt Cl function C-41
ClFlushCommand Cl function C-44
ClGetDialogInstances Cl function C-45
ClGetDialogLastInstanceId Cl function C-46
ClGetDialogMaxInstances Cl function C-47
ClGetDialogNumCurInstances Cl function C-48
ClGetFloatStringVal Cl function C-49
ClGetIntegerStringVal Cl function C-50
ClInitialize Cl function C-51
ClLaunchDialog Cl function C-52
ClosedCb callback 5-9
called when you close, cancel or confirm a DialogBox 5-9
DialogBox 5-9
TopLevel 5-9
ClRegisterDialog Cl function C-53
ClRegisterDialogMaxInstances Cl function C-54
ClRequest Cl function C-56
ClSendCCommand Cl function C-65
ClSendErrorCode Cl function C-66
ClSendErrorCString Cl function C-67
ClSendErrorHliString Cl function C-68
ClSendWarning Cl function C-69
ClSetPrompt Cl function C-70
clwrap.h
file C-1
Color
Name 5-30
Specifier(cust) 5-38
Color attribute
HLS(cust) 5-23
ColorCategory
HLS(cust) 5-23
Column Alignments attribute 5-10
array of text alignments 5-10
ScrollMatrix 5-10
ColumnLabel attribute 5-10
ScrollMatrix 5-10
text for the labels above a column 5-10
ColumnLabels attribute 5-10
array of labels drawn above each column 5-10
ScrollMatrix 5-10
ColumnsNum attribute 5-10
number of columns in the cell 5-10
ScrollMatrix 5-10
ColumnsWide attribute 5-11
Field 5-11
specifies the number of columns that a field will occupy in an aligning form 5-11
ColumnWidths
array of widths, one for each column, specifying visible width of the cells in the column 5-10
ColumnWidths attribute 5-10
ScrollMatrix 5-10
Command
cadds 4-4
share_cvact 4-4

Command Argument Box
  graphic objects 1-5
Command Argument Box(CAB)
  creating 2-2
  rearranging objects 2-4
CommandBox
  ApplyButton 5-4
  ApplyCb 5-4
  ApplyLabel(cust) 5-5
  ButtonParent 5-6
  CancelCb 5-6
  CancelLabel 5-7
  OkLabel(cust) 5-31
  VisibleWhenManaged(cust) 5-42
  WindowTitle(cust) 5-43
CommandBox (CAB) class 6-14
  gets arguments from the user 6-14
CommandEntry
  Accelerator(cust) 5-3
  ArgumentBox 5-5
CommandEntry class
  Class
    Command Entry 6-16
  displays a CAB 6-16
CommandOn(cust) attribute 5-11
  ApplicationDesktop 5-11
  indicates whether the command area is to be visible 5-11
Communicating with CADDS
  CL functions C-1
CompleteCb callback 5-11
  called when focus leaves the object 5-11
  Fillin 5-11
  OptionList 5-11
  Paragraph 5-11
  ScrollList 5-11
CompleteCellCb callback 5-12
  called after a cell has been edited 5-12
  ScrollMatrix 5-12
ConstantHeight attribute 5-12
  AligningForm 5-12
  indicates that all rows are of the same height, if TRUE 5-12
ConstantWidth attribute 5-13
  AligningForm 5-13
  indicates that all columns have the same width, if TRUE 5-13
Container
  Children 5-8
  ChildrenNum 5-8
  Framed 5-21
Container class 6-17
  object that can have children 6-17
Contents(cust) attribute 5-13
  EntryList 5-13
  specifies contents of an EntryList 5-13
ContentsNum attribute 5-13
  EntryList 5-13
  number of entries in the contents 5-13
CR Function
  invoking from CADDS C-5
CR Process 1-6
CR process
  active variables C-7
crdllfdef.h
  header file
    dynamic loading
      Windows NT 4-7
Creating
  Command Argument Box(CAB) 2-2
  interface 2-2
  objects 2-2
  shared library 1-5, 4-2
    libUserCr.so 4-2
  simple Dialog Box 3-24
CurrentTool(cust) attribute 5-13
  ApplicationDesktop 5-13
  ToolEntry which is the current tool in use 5-13
Cursor UiObject 5-13
  ToolEntry 5-13
  used in the presentation area from the time that a tool is activated until it is deactivated 5-13
CursorPosition attribute 5-14
  Fillin 5-14
  moves cursor, if set 5-14
  Paragraph 5-14
  ScrollList 5-14
cust superscript 4-12
Custom CVact menus
  dynamic loading 1-3
  porting to Windows NT 1-3
Customizing
  Advanced Customization Tool 1-2
  attributes 3-30
    resources 3-30
  CADDS applications 1-5
  how to override 4-15

menu 4-12
sample .CR file 4-15
objects 3-30
color 3-31
attributes 3-34
font 3-31
attributes 3-33
resources 3-30
sensitivity 3-30
visibility 3-30
CVact xlvii
exiting 1-15
getting started 1-7
trademark window 1-7
CVact Classes
IntegerFillin
calculator 6-32
CVact menu 4-4
behavior
when iconized
when deiconized 4-16
fcode
specifying 4-10
icode
specifying 4-10
specifying Final Code (fcode) 4-10
specifying Initial Code (icode) 4-10
CVUI
executable 4-3
standalone testing
shared library 4-3
CVUIOVERRIDEPATH environment 4-15

D

DatasetWindow
StatusArea 5-39
DataSetWindow class 6-18
provides window for the user 6-18
DeactivateCb attribute 5-14
DeactivateCb callback
issued when the user drops a tool 5-14
ToolEntry 5-14
DecimalPlaces(cust) attribute 5-14
FloatFillin 5-14
number of places to the right of the decimal that are to be shown when values are converted from FloatValue 5-14
DefaultActionCb callback 5-14
called when an item is invoked 5-14
ScrollList 5-14
DefaultBoolean(cust) attribute 5-15
default value 5-15
Toggle 5-15
DefaultButton attribute 5-15
DialogBox 5-15
DefaultButtoon attribute
one of the child push buttons that is the default push button 5-15
DefaultFloatValue(cust) attribute 5-15
automatically sets the DefaultString value, when set 5-15
FloatFillin 5-15
DefaultIntegerValue(cust) attribute 5-15
IntegerFillin 5-15
sets the DefaultString value when set 5-15
DefaultItems attribute 5-16
default list of chosen data 5-16
ScrollList 5-16
DefaultItemsNum attribute 5-16
number of items in DefaultItems array 5-16
ScrollList 5-16
DefaultObject attribute 5-16
name of the default toggle choice 5-16
RadioSet 5-16
DefaultString(cust) attribute 5-16
default text 5-16
Fillin 5-16
FromMessage convenience functions 5-16
Paragraph 5-16
DefaultStringNum attribute 5-17
Fillin 5-17
number of characters in the Default String 5-17
DefaultValue(cust) attribute
a string showing a default value 5-17
Attribute
DefaultValue(cust) 5-17
OptionList 5-17
ScrollList 5-17
Defective attribute 5-17
Fillin 5-17
Defective attriibute
indicates that the contents of the field have been accepted, if TRUE 5-17
Definer class variable 5-17
Entry 5-17
holds the address of the application entry definer 5-17

Dependency 3-6
attributes
setting 3-8
expressions 3-7
Dependency attribute 5-17
specifies which attribute is affected by the DependencyExpression 5-17
UiObject 5-17
DependencyExpression attribute 5-18
character string containing the dependency expression 5-18
Field 5-18
DependencyType attribute 5-18
Field 5-18
tells whether the dependency expression controls Sensitive, VisibleWhenManaged, or Managed attribute 5-18
Description
Icon Converter 3-16
Design Rule Customization
dialog box 4-13
example of overriding 4-15
Design Rule Customization tool 4-12
Designing a GUI 1-2
UiObjects 1-2
DestroyCb callback 5-18
invoked when object is about to be destroyed 5-18
UiObject 5-18
Dialog Box 1-2
CADDS LAF (Look and Feel) 1-2
Design Rule Customization 4-13
Motif LAF 1-2
saving 2-14
simple
creating 3-24
UiObjects 1-16
DialogBox
ClosedCb 5-9
DefaultButton 5-15
EditorFocusChangeCb 5-19
EditorIcon 5-19
EditorInvokedCb 5-20
EditorRescindCb 5-20
HeightSpecified 5-22
Modal 5-29
Parent 5-32
VisibleWhenManaged(cust) 5-42
WidthSpecified 5-43
WindowTitle(cust) 5-43
XSpecified 5-44
YSpecified 5-45
DialogBox class 6-19
top-level window 6-19
DirectActionEntry
Accelerator(cust) 5-3
DirectActionEntry class 6-20
invokes immediate action 6-20
Directory
Example 4-9
Documentation, printing from Portable Document Format (PDF) file 1
DropCb callback 5-18
distinguishes the field visually indicating that dropping is permitted, if non-null 5-18
Field 5-18
Dynamic loading
custom CVact menus 1-3
Windows NT
custom CVact menus 4-8
header file
crdllfdef.h 4-7
Dynamically loaded library
libUserCr.dll 4-7

E

Editing
object properties
multiple objects 2-13
the Message and Callback Editors 2-10
using the Object Property Editor 2-10
setting attributes 2-10
Editor attribute 5-19
Fillin 5-19
FloatFillin 5-19
IntegerFillin 5-19
provides the icon used for the Editor Icon 5-19
EditorAttributes attribute 5-19
Fillin 5-19
FloatFillin 5-19
IntegerFillin 5-19
set of attributes affecting the icon and behavior of EditorDialogBox 5-19
EditorDialogBox class 6-21
Associating the EditorDialogBox with a Fillin

Field 6-21
helps the user fill or edit the contents of a fillin field 6-21
EditorFocusChangeCb callback 5-19
DialogBox 5-19
issued when an associated filllin field is the focus of attention 5-19
EditorIcon UiObject
DialogBox 5-19
supplied if the dialog box is to be an EditorDialogBox 5-19
editorIcon UiObject 5-19
EditorInvokedCb callback 5-20
DialogBox 5-20
invoked when you press the Editor Button in an associated field 5-20
EditorRescindCb callback 5-20
DialogBox 5-20
issued when a dialog box containing an associated fillin field is closed 5-20
Entry
ActivateCb 5-3
Definer 5-17
Name 5-30
TextValue(cust) 5-40
Entry attribute 5-20
Menu 5-20
shows TextValue 5-20
Entry class
defines the attributes of its derived classes 6-22
EntryList
Contents(cust) 5-13
ContentsNum 5-13
Inline(cust) 5-24
Pointer 5-32
EntryList class 6-23
container object 6-23
Environment variable
CVUIOVERRIDEPATH 4-15
SHLIB_PATH 4-6
UI_LOOK_AND_FEEL 4-4
ErrorBox class 6-24
informs the user of an error 6-24
Example
Icon Converter 3-16
Example directory 4-9
Executable
CVUI 4-3
Exiting
CVact 1-15

F

FailureBox class 6-25
informs the user of irrecoverable problem 6-25
Field
ColumnsWide 5-11
DependencyExpression 5-18
DependencyType 5-18
DropCb 5-18
Framed 5-21
IntializeCb 5-24
Prefilled 5-32
RowsHigh 5-37
StartColumn 5-39
StartRow 5-39
Field values
CVact menu
iconized
deiconized 4-16
File
.act 4-2
.caddsrc 4-4
.CR 4-15
appinitU.cxx 4-4
clwrap.h C-1
header
crdllfdef.h
Windows NT 4-7
libUserCr.so 4-3
Makefile 1-6
object (.o) 4-2
X resource 4-13
Fillin
CompleteCb 5-11
CursorPosition 5-14
DefaultString(cust) 5-16
DefaultStringNum 5-17
Defective 5-17
Editor 5-19
EditorAttributes 5-19
FontCategory 5-21
InputPermitted 5-24
IsInputOkCb 5-25
MaxLength 5-27
MaxVisibleLength 5-27
MinVisibleLength 5-29
MinVisibleLines 5-29
RememberedString 5-35
RememberedStringNum 5-35

TextString 5-40
TextStringNum 5-40
fillin 5-35
Fillin class 6-26
allows single line text input 6-26
FloatFillin
DefaultFloatValue(cust) 5-15
Editor 5-19
EditorAttributes 5-19
FloatValue 5-20
IsInputOkCb 5-25
MaxFloatValue(cust) 5-27
MaxLength 5-27
MinFloatValue(cust) 5-28
FloatFillin class 6-28
allows single line text input of a floating point value 6-28
FloatFilllin
DecimalPlaces(cust) 5-14
FloatValue attribute 5-20
FloatFillin 5-20
floating-point representation of the TextString 5-20
MaxFloatValue 5-20
MinFloatValue 5-20
Focus
keyboard 5-12
mouse 5-12
FontCategory
Name 5-30
Specifier(cust) 5-38
FontCategory attribute 5-21
Fillin 5-21
Label 5-21
OptionList 5-21
Paragraph 5-21
ScrollList 5-21
specifies the font category to be used to display text in an object 5-21
Foreground attribute 5-21
BuildingBlock 5-21
foreground color category for an object 5-21
Framed attribute 5-21
Container 5-21
Field 5-21
NonAligningForm 5-21
RadioSet 5-21
specifies whether to display a frame around an object 5-21
FromMessage convenience functions
ApplyLabel(cust) 5-5
AssistLIne(cust) 5-5
CancelLabel 5-7
DefaultString(cust) 5-16
IconifiedTitle(cust) 5-23
OkLabel(cust) 5-31
Prompt 5-33
ResetLabel 5-36
TextString 5-40
TextValue(cust) 5-41
Title(cust) 5-41
WindowTitle(cust) 5-43
Function
Ui....CreateFromVStatus A-33
UiActivate A-10
UiAddColumns A-11
UiAddEntry A-12
UiAddItem A-13
UiAddItems A-15
UiAddRows A-17
UiAddSeparator A-19
UiAligningFormCreate A-20
UiAppDtCreate A-21
UiApplicationStateCreate A-22
UiApplyPulldownCreate A-23
UiAssistOn A-24
UiChooseItem A-25
UiClose A-26
UiCommandBoxCreate A-27
UiCommandEntryCreate A-29
UiCompletion A-31
UiConfirm A-32
UiDatasetWindowCreate A-34
UiDeleteColumns A-35
UiDeleteEntry A-36
UiDeleteItem A-37
UiDeleteRows A-38
UiDeleteSeparator A-39
UiDialogBoxCreate A-41
UiDirectEntryCreate A-42
UiEditCell A-43
UiEditorDialogBoxCreate A-44
UiEndBusy A-45
UiEntryLIstCreate A-46
UiErrorBoxCreate A-47
UiFailureBoxCreate A-48
UiFillinCreate A-49
UiFillinStrlen A-50

UiFloatFillinCreate A-51
UiGet... A-52
UiGet...Cb A-53
UiGetCell A-55
UiGetCellColor A-56
UiGetColumnColors A-57
UiGetColumnLabel A-58
UiGetDependency A-59
UiGetEntry A-60
UiGetEntryLabel A-61
UiGetEntryRequired A-62
UiGetFocus A-63
UiGetGeometry A-64
UiGetGeometrySpecified A-65
UiGetInstanceId A-66
UiGetItem A-67
UiGetItemSensitive A-68
UiGetObjectByName A-69
UiGetPositionInDesktop A-71
UiGetResetInProgress A-72
UiGetRowColors A-73
UiGetRowColumn A-74
UiGetRowLabel A-75
UiGetRowLabels A-76
UiIdentificationBoxCreate A-77
UiInformationBoxCreate A-78
UiIntegerFillinCreate A-79
UiIsDefaultButton A-80
UiLabelCreate A-81
UiNonAligningFormCreate A-82
UiObtainEntry A-83
UiOptionListCreate A-84
UiPaletteCreate A-85
UiParagraphCreate A-86
UiParagraphInsertText A-87
UiParagraphMoveCursor A-88
UiPromptOn A-90
UiPushButtonCreate A-91
UiQueryItem A-92
UiQueryResultBoxCreate A-94
UiQuestionBoxCreate A-95
UiRadioScrollListCreate A-96
UiRadioSetCreate A-97
UiReset A-98
UiResetCab A-99
UiScrollListCreate A-100
UiScrollMatrixCreate A-101
UiSeparatorCreate A-102
UiSet... A-103
UiSet...Cb A-105
UiSet...FromMessage A-120
UiSetBorderWidthForSldoffDlgBox A-104
UiSetCell A-107
UiSetCellColor A-108
UiSetCellValues A-109
UiSetChosen A-110
UiSetColumnAlignments A-111
UiSetColumnColors A-112
UiSetColumnLabel A-113
UiSetColumnLabels A-114
UiSetDefaultButtonWithSThickness A-115
UiSetDependency A-116
UiSetEditorAttributes A-117
UiSetEntryLabel A-118
UiSetEntryRequired A-119
UiSetGeometrySpecified A-121
UiSetInstanceId A-122
UiSetItemSensitive A-123
UiSetMatrix A-124
UiSetRowColor A-126
UiSetRowColumn A-127
UiSetRowLabel A-128
UiSetRowLabels A-129
UiSetSpecifiedPosition A-132
UiSetSpecifiedSize A-133
UiSlideOffDialogBoxCreate A-130
UiSlideOffEditorDialogBoxCreate A-131
UiSpecifierCreate A-134
UiStartBusy A-135
uiToggleCreate A-136
UiToggleEntryCreate A-137
UiWarningBoxCreate A-138
UiWorkingBoxCreate A-139

G

Generating
  shared library 4-2
Graphic objects 1-4
  Command Argument Box (CAB) 1-5
GUI
  designing 1-2
Gutter attribute 5-21
  Aligning Form 5-21
Gutter attriubte
  space between children of an aligning form 5-21

H

Height attribute 5-22
  BuildingBlock 5-22
  current height of the building block 5-22
HeightSpecified attribute 5-22
  applications's preference for the height of the box 5-22
  BuildingBlock 5-22
  DialogBox 5-22
HelpContext attribute 5-22
  BuildingBlock 5-22
  passed to the Help Subsystem if the user asks for help 5-22
  UiObject 5-22
Hierarchies 3-2
  instancing 3-2
  parenting 3-2
Hierarchy
  class attributes 6-3
  class attributes and UI Functions 6-3
  inheritance tree 6-3
  UI functions 6-3
Hli function
  HliCCharToChar B-3
  HliCharToCChar B-9
  HliCNStringToString B-4
  HliCStrdup B-5
  HliCStringFromString B-6
  HliCStringToString B-7
  HliCStrndup B-8
  HliExternNStringToString B-10
  HliExternStringFromString B-11
  HliExternStringToString B-12
  HliFromCString B-13
  HliFromExternString B-14
  HliShutdown B-15
  HliStringNToCString B-16
  HliStringNToExternString B-17
  HliStringToCString B-18
  HliStringToExternString B-19
  HliStringVFormat B-20
HliCCharToChar Hli function B-3
HliCharToCChar Hli function B-9
HliCNStringToString Hli function B-4
HliCStrdup Hli function B-5
HliCStringFromString Hli function B-6
HliCStringToString Hli function B-7
HliExternStringFromString Hli function B-11
HliExternStringToString Hli function B-12
HliFromCString Hli function B-13
HliFromExternString Hli function B-14
HliShutdown Hli function B-15
HLIString
  utility 3-22
HliStringNToCString Hli function B-16
HliStringNToExternString Hli function B-17
HliStringToCString Hli function B-18
HliStringToExternString Hli function B-19
HliStringVFormat Hli function B-20
HliStrndup Hli function B-8
HLS(cust) attribute
  a color attribute specified in Hue/Lightness/Saturation values 5-23
  ColorCategory 5-23
  Hue/Lightness/Saturation values 5-23
HLS(cust) color attribute 5-23
Hue/Lightness/Saturation values
  HLS(cust) attribute 5-23

I

Icon Converter 3-16
  description 3-16
  example 3-16
  syntax 3-16
Icon Editor 3-14
Icon(cust) UiObject 5-23
  (Label) 5-23
  icon displayed on an object 5-23
IconifiedIcon(cust) UiObject 5-23
  icon displayed when the object is iconified 5-23
  TopLevel 5-23
IconifiedTitle(cust)
  title displayed under the icon when object is iconified 5-23
IconifiedTitle(cust) attribute
  FromMessage convenience functions 5-23
IconifiedTitle(cust) UiObject 5-23
  TopLevel 5-23
Icons
  converting 3-16
  editing 3-14
  examples 3-12
  specifying 3-11

using 3-11
IdentificationBox
UserText(cust) 5-41
IdentificationBox class 6-30
identifies an application 6-30
InformationBox class
displays information message 6-31
Inheritance tree
how to use 6-3
InitializeCb callback 5-24
Field 5-24
issued when object is going to be reset or made visible 5-24
ScrollList 5-24
Inline(cust) attribute 5-24
EntryList 5-24
signifies that entries in a list are to appear inline in the current menu 5-24
InputPermitted attribute 5-24
Fillin 5-24
Paragraph 5-24
permits input, if TRUE 5-24
InsCircleCab menu 4-4
Instance
Children 5-8
Parent 5-32
Instancing
hierarchy 3-2
IntegerFillIn
IsInputOkCb 5-25
IntegerFillin
DefaultIntegerValue(cust) 5-15
Editor 5-19
EditorAttributes 5-19
IntegerValue 5-25
MaxIntegerValue(cust) 5-27
MaxLength 5-27
MinIntegerValue(cust) 5-28
IntegerFillin class 6-32
allows single line text input of an integer value 6-32
calculator 6-32
inverse functions 6-32
IntegerValue attribute 5-25
integer representation of the TextString 5-25
IntegerFillin 5-25
Interface
changing 3-1
creating 2-2
customizing
Advanced Customization Tool 1-2
loading into the design tool 2-15
removing 2-15
saving 2-14
Invoking
CR function from CADDS C-5
IsEditableCb callback 5-25
called immediately before a cell is to be edited 5-25
ScrollMatrix 5-25
IsInputOkCb callback 5-25
Fillin 5-25
FloatFillin 5-25
IntegerFillin 5-25
issued when contents of an object change due to entry or deletion of a character 5-25
Paragraph 5-25
ScrollMatrix 5-25
Item
ChildrenNum 5-8
Item attribute 5-26
OptionList 5-26
ScrollList 5-26
text value of an item in a scroll list or option list 5-26
ItemsNum attribute 5-26
number of items in a scroll list 5-26
OptionList 5-26
ScrollList 5-26

K

Keyboard focus 5-12

L

Label
Alignment 5-3
FontCategroy 5-21
TextValue(cust) 5-40
Label class
displays text value 6-34
LabelFontCategory
font category used for displaying row and columns labels 5-26
LabelFontCategory attribute 5-26
ScrollMatrix 5-26
Layout

changing 3-1
managing 3-3
options
specifying 3-10
LD_LIBRARY_PATH 4-3
Library
libUserCr 4-5
sharing 4-2
libUserCr library 4-5
libUserCr.dll
dynamically loaded library 4-7
libUserCr.so
creating 4-2
file 4-3
shared library 4-2

M

Manage attribute 5-26
BuildingBlock 5-26
indicates that this building block participates in geometry management, if TRUE 5-26
Managing
Geometry 3-3
Mask
UI_SHIFT_MOD_MASK 5-9, 5-15
MaxFloatValue
FloatValue 5-20
MaxFloatValue(cust) attribute 5-27
FloatFillin 5-27
maximum value permitted for FloatValue 5-27
MaxIntegerValue(cust) attribute 5-27
IntegerFillin 5-27
maximum value permitted for IntegerValue 5-27
MaxLength attribute 5-27
Fillin 5-27
FloatFillin 5-27
IntegerFillin 5-27
maximum length, in characters, permitted to enter 5-27
Paragraph 5-27
MaxVisibleLength attribute 5-27
Fillin 5-27
maximum number of visible characters preferred 5-27
Paragraph 5-27
MaxVisibleLilnes attribute 5-28
MaxVisibleLines attribute
maximum number of visible lines preferred 5-28
Paragraph 5-28
Menu
customizing 4-12
sample .CR file 4-15
CVact 4-4
Entry 5-20
InsCircleCab 4-4
Menus in CADDS
preparing to use 4-3
sample menu
using 4-9
using 4-4
Message and Callback Editors 2-10
MinFloatValue
FloatValue 5-20
MinFloatValue(cust) attribute 5-28
FloatFillin 5-28
minimum value permitted for FloatValue 5-28
MinIntegerValue(cust) attribute 5-28
IntegerFillin 5-28
minimum value permitted for IntegerValue 5-28
MinVisibleColumns attribute 5-28
number of visible columns preferred 5-28
ScrollMatrix 5-28
MinVisibleLength attribute 5-29
Fillin 5-29
Paragraph 5-29
preferred minimum number of visible characters 5-29
MinVisibleLines
preferred number of items to be shown 5-29
MinVisibleLines attribute 5-29
Fillin 5-29
Paragraph 5-29
MinVisibleRows attribute 5-29
preferred number of rows that should be visible at a time 5-29
ScrollMatrix 5-29
MinVisinbleLInes attribute
ScrollList 5-29
Modal attribute 5-29
DialogBox 5-29
indicates that the user cannot work elsewhere in the application without dismissing the dialog box 5-29
Modst 5-8, 5-15
MOTIF 4-4

Mouse button
adjust 1-14
menu 1-14
select 1-14
MouseEventCb callback 5-30
invoked when a mouse event occurs 5-30
ToolEntry 5-30
MouseEventMask bit mask 5-30
ToolEntry 5-30
MouseEventMask mask
defines the class of mouse events which triggers a mouse event callback 5-30
Multiple objects
editing
properties 2-13
selecting 2-9

N

Name object instance 5-30
absolute name of this object instance 5-30
ApplicationDesktop 5-30
ApplicationState 5-30
Color 5-30
Entry 5-30
FontCategory 5-30
UiObject 5-30
NonAligningForm
Framed 5-21
Title(cust) 5-41
NonAligningForm class 6-35
used when visual alignment into rows and columns is not preferred 6-35
NumColumns(cust) attribute 5-31
AligningForm 5-31
Palette 5-31
tells how many columns are to be used for position of the children 5-31

O

Object file (.o) 4-2
Object instance
Name 5-30
Object Property Editor
changing object to be edited 2-9
editing
object properties 2-10
setting attributes 2-10
using to rearrange objects 2-7
Objects
adding 2-3
changing
in the Object Property Editor
to be edited 2-9
creating
from the palette 2-2
customizing 3-30
multiple
editing properties 2-13
properties
editing 2-10
rearranging
in a Command Argument Box(CAB) 2-4
using the Object Property Editor 2-7
reparenting 2-4
tree of objects 2-4
selecting
multiple 2-9
sensitivity 3-5
visibility 3-4
OkLabel(cust) attribute 5-31
CommandBox 5-31
FromMessage convenience functions 5-31
label used for the OK button 5-31
Option Lists 2-12
setting attributes 2-12
OptionList
ChosenCb 5-8
CompleteCb 5-11
DefaultValue(cust) 5-17
FontCategory 5-21
Item 5-26
ItemsNum 5-26
Remembered 5-34
TextValue(cust) 5-40
OptionList class
presents non-scrollable menu items for selection 6-36
Orientation attribute 5-31
Separator 5-31
tells whether the separator should be horizontal or vertical 5-31
OtherData attribute 5-31
open space for the application to attach any information it prefers to the object

instance 5-31
UiObject 5-31

P

Palette
NumColumns(cust) 5-31
Source(cust) 5-38
UIObjects 1-8
adding more objects 2-3
Command Argument Box(CAB)
creating 2-2
UiObjects
creating 2-2
Palette class
row column container for graphic or textual menu entries 6-37
Paragraph
CompleteCb 5-11
CursorPosition 5-14
DefaultString(cust) 5-16
FontCategory 5-21
InputPermitted 5-24
IsInputOkCb 5-25
MaxLength 5-27
MaxVisibleLength 5-27
MaxVisibleLines 5-28
MinVisibleLength 5-29
MinVisibleLInes 5-29
RememberedString 5-35
RememberedStringNum 5-35
TextString 5-40
TextStringNum 5-40
Paragraph class 6-38
allows scrolling of the multi-line text 6-38
scrollbars 6-38
Parent instance 5-32
ApplicationDesktop 5-32
BuildingBlock 5-32
DialogBox 5-32
Parenting
hierarchy 3-2
Partial syntax
attribute 5-2
PercentDrawn attribute 5-32
percent of the available spanning distance that should be drawn 5-32
Separator 5-32
Platform specific information
shared library 4-5
Pointer attribute 5-32
EntryList 5-32
Porting
from UNIX to Windows NT
custom CVact menus 1-3
PositionInDesktop attribute 5-32
BuildingBlock 5-32
position of the Building Block relative to the Application Desktop 5-32
Prefilled attribute 5-32
Field 5-32
tells whether the field is to have its prefilled value obtained from default, remembered from invocation, or recomputed each time the field is displayed 5-32
Printing documentation from Portable Document Format (PDF) file 1
Prompt string 5-33
ApplicationDesktop 5-33
FromMessage convenience functions 5-33
tells the user what to do next 5-33
PromptOn(cust) attribute 5-33
ApplicationDesktop 5-33
indicates whether the prompt area is to be made visible 5-33
PropertiesCb callback 5-33
called when the user asks to edit Tool properties 5-33
ToolEntry 5-33
Property Editor 1-13
Pulldown 1-10
Edit 1-10
File 1-10
Interfaces 1-12
Options 1-12
Tools 1-13
View 1-11
Push button
AlternateIcon(cust) 5-4
PushButton
TextValue(cust) 5-40
Pushbutton
Alignment 5-3
ResetCb 5-36
ResetLabel 5-36
PushButton class
provides easy input, specifies action 6-40

PushedCb callback 5-33
  invoked when the button is pushed 5-33

Q

QueryResultBox class 6-42
  displays information which is the result of a query 6-42
QuestionBox class 6-43
  used to ask the user yes/no question 6-43

R

RadioBehavior attribute 5-34
  only a single choice is permitted, if TRUE 5-34
  ScrollList 5-34
RadioSet
  ChosenCb 5-8
  ChosenObject 5-9
  DefaultObject 5-16
  Framed 5-21
  RememberedObject 5-35
RadioSet class
  contains toggle buttons where only a single selection is allowed 6-44
RemeberedItems attribute
  ScrollList 5-35
Remembered attribute 5-34
  the remembered value 5-34
remembered attribute
  OptionList 5-34
RememberedBoolean attribute 5-34
  remembered value from the last time the field was visible and confirmed 5-34
  Toggle 5-34
RememberedItems attribute
  remembered list of chosen data 5-35
RememberedItemsNum attribute 5-35
  number of items in RememberedItems 5-35
  ScrollList 5-35
RememberedObject attribute
  RadioSet 5-35
  remembered choice from the last time the set was visible and confirmed 5-35
RememberedString attribute
  5-35
  Paragraph 5-35
  remembered text from the last time the field was visible and confirmed 5-35
RememberedStringNum attribute
  Fillin 5-35
  number of characters in the Default String 5-35
  Paragraph 5-35
RememberedStrinNum attribute 5-35
Rerranging objects
  with the Object Property Editor 2-7
ResetCb callback 5-36
  called when the Reset button is pressed 5-36
  Pushbutton 5-36
ResetLabel attribute 5-36
  FromMessage convenience functions 5-36
  Pushbutton 5-36
  text label for the Reset button 5-36
RGB attribute
  BuildingBlock 5-34
  color type of red, green, and blue 5-34
RowColumn attribute 5-36
  any UiObject inside an AlignForm has a RowColumn attribute 5-36
  BuildingBlock 5-36
RowLabel attribute 5-37
  one of the RowLabels 5-37
  ScrollMatrix 5-37
RowLabels attribute 5-37
  an array of HliStrings that make up the labels to be drawn to the left of each row 5-37
  ScrollMatrix 5-37
RowsHigh attribute 5-37
  Field 5-37
  number of rows that a field occupies 5-37
RowsNum attribute 5-37
  number of rows in a ScrollMatrix object 5-37
  ScrollMatrix 5-37

S

Saving
  Dialog Box 2-14
  interface 2-14
    Dialog Box 2-14
Script
  share_cvact 4-5
  share_cvact.pl 4-7
ScrollList
  Chosen 5-8
  ChosenCb 5-8
  ChosenItems 5-9

ChosenItemsNum 5-9
CompleteCb 5-11
CursorPosition 5-14
DefaultActionCb 5-14
DefaultItems 5-16
DefaultItemsNum 5-16
DefaultValue(cust) 5-17
FontCategroy 5-21
InitializeCb 5-24
Item 5-26
ItemsNum 5-26
MinVisibleLines 5-29
RadioBehavior 5-34
RememberedItems 5-35
RememberedItemsNum 5-35
ScrollList class
presents entries that can be scrolled for selection 6-45
ScrollMatrix
Cell 5-7
Column Alignments 5-10
ColumnLabel 5-10
ColumnLabels 5-10
ColumnsNum 5-10
ColumnWidths 5-10
CompleteCellCb 5-12
IsEditableCb 5-25
IsInputOkCb 5-25
LabelFontcategory 5-26
MinVisibleColumns 5-28
MinVisibleRows 5-29
RowLabel 5-37
RowLabels 5-37
RowsNum 5-37
TextFontCategory 5-40
ScrollMatrix class 6-47
two-dimensional array of text strings for editing 6-47
Selected Objects Menu 1-13
Selecting
multiple objects 2-9
Sending
coding error messages to CADDS C-7
Sending commands to CADDS C-4
Sensitive(cust) attribute 5-37
BuildingBlock 5-37
displays the object normally, if TRUE 5-37
UiObject 5-37
SensitiveNow attribute 5-38
BuildingBlock 5-38
tells whether the building block is currently sensitive 5-38
Sensitivity
customizing 3-30
Separator
Orientation 5-31
PercentDrawn 5-32
Separator class 6-49
separates fields in a display 6-49
Setting
dependency attributes 3-8
Setting attributes
Message and Callback Editors 2-10
seeing what has changed 2-13
Text Editor 2-11
using the Object Property Editor 2-10
using the Option lists 2-12
share_cvact command 4-4
share_cvact script 4-5
share_cvact.pl script 4-7
Shared library
creating 1-5, 4-2
generating 4-2
libUserCr.dll 4-7
libUserCr.so 4-2
platform specific information 4-5
standalone testing
using CVUI 4-3
using CVUI executable 4-3
SHLIB_PATH environment variable 4-6
SlideOffEditorDlgBox class
associating with Fillin field 6-50
helps fill out or edit the contents of a fillin field 6-50
Slider class
provides facility to enter a value by a slide operation 6-52
Source(cust) attribute 5-38
Palette 5-38
passed to the EntyDefiner procedure for an entry in a Palette or Menu 5-38
UiMenu 5-38
Specifier(cust) attribute 5-38
Color 5-38
color specifier string 5-38
FontCategory 5-38
SpecifierEntry
Accelerator(cust) 5-3

SpecifierEntry class 6-53
  displays a specifier in the specifier bar 6-53
Specifying
  icons 3-11
  layout options 3-10
StartColumn attribute 5-39
  Field 5-39
  specifies that a field prefers to start at a particular column location within an AligningForm parent 5-39
Starting
  CVact 1-7
StartRow attribute 5-39
  Field 5-39
  specifies that a field intends to start at a particular row location within an AligningForm parent 5-39
State attribute 5-39
  ApplicationState 5-39
  tells whether state is TRUE or FALSE 5-39
StatusArea attribute 5-39
  allows status fields to be added to an AligningForm 5-39
  DatasetWindow 5-39
Sticky attribute 5-39
String
  Prompt 5-33
Superscript
  cust 4-12
Syntax
  attribute
    partial 5-2
  Icon Converter 3-16

T

Text Editor 2-11
  setting attributes 2-11
TextFontCategory attribute 5-40
  font category used for the text displayed in cells 5-40
  ScrollMatrix 5-40
TextString attribute 5-40
  current contents of the field 5-40
  Fillin 5-40
  FromMessage convenience functions 5-40
  Paragraph 5-40
TextStringNum attribute 5-40
  Fillin 5-40
  number of characters in the TextString 5-40
  Paragraph 5-40
TextValue(cust) attribute
  Entry 5-40
  FromMessage convenience functions 5-41
  Label 5-40
  PushButton 5-40
  string value displayed 5-40
textValue(cust) attribute
  OptionList 5-40
  Toggle 5-40
Title(cust) attribute 5-41
  AligningForm 5-41
  FromMessage convenience functions 5-41
  NonAligningForm 5-41
  title of the form 5-41
Toggle
  AlternateIcon(cust) 5-4
  Checked(cust) 5-7
  ChosenCb 5-8
  DefaultBoolean(cust) 5-15
  RememberedBoolean 5-34
  TextValue(cust) 5-40
Toggle class
  two-state button that the user can toggle 6-54
ToggleEntry
  Accelerator(cust) 5-3
ToggleEntry class
  special case of direct action entries 6-55
Tool
  Advanced Customization 1-2
  Design Rule Customization (DRC) 4-12
ToolEntry
  AbortCb 5-2
  Accelerator(cust) 5-3
  Cursor 5-13
  DeactivateCb 5-14
  MouseEventCb 5-30
  MouseEventMask 5-30
  PropertiesCb 5-33
TopLevel
  ClosedCb 5-9
  IconifiedIcon(cust) 5-23
  IconifiedTitle(cust) 5-23
  WindowTitle(cust) 5-43
TopLevel class
  building block under the control of the window manager 6-56
Tree
  of objects

reparenting objects 2-4

U

UI Callback Editor 3-23
Ui Function
  UiPostCab A-89
Ui....CreateFromVStatus function A-33
UI_LOOK_AND_FEEL environment variable 4-4
UI_SHIFT_MOD_MASK 5-15
UI_SHIFT_MOD_MASK mask 5-9
UiActivate function A-10
UiAddColumns function A-11
UiAddEntry function A-12
UiAddItem function A-13
UiAddItems function A-15
UiAddRows function A-17
UiAddSeparator function A-19
UiAligningFormCreate function A-20
UiApplicationStateCreate function A-22
UiApplyPulldownCreate function A-23
UiAppStCreate function A-21
UiAssistOn function A-24
UiCallback
  WindowActivateCb 5-43
UiChooseItem function A-25
UiClose function A-26
UiCommandBoxCreate function A-27
UiCommandEntryCreate function A-29
UiCompletion function A-31
UiConfirm function A-32
UiDatasetWindowCreate function A-34
UiDeleteColumns function A-35
UiDeleteEntry function A-36
UiDeleteItem function A-37
UiDeleteRows function A-38
UiDeleteSeparator function A-39
UiDialogBoxCreate function A-41
UiDirectEntryCreate function A-42
UiEditCell function A-43
UiEditorDialogBoxCreate function A-44
UiEndBusy function A-45
UiEntryListCreate function A-46
UiErrorBoxcreate function A-47
UiFailureBoxCreate function A-48
UiFillinCreate function A-49
UiFillinStrlen function A-50
UiFloatFillinCreate function A-51
UiGet... function A-52
UiGet...Cb function A-53
UiGetCell function A-55
UiGetCellColor function A-56
UiGetColumnColors function A-57
UiGetColumnLabel function A-58
UiGetDependency function A-59
UiGetEntry function A-60
UiGetEntryLabel function A-61
UiGetEntryRequired function A-62
UiGetFocus function A-63
UiGetGeometry function A-64
UiGetGeometrySpecified function A-65
UiGetInstanceId function A-66
UiGetItem function A-67
UiGetItemSensitive function A-68
UiGetObjectByName function A-69
UiGetPositionInDesktop function A-71
UiGetResetInProgress function A-72
UiGetRowColors function A-73
UiGetRowColumn function A-74
UiGetRowLabel function A-75
UiGetRowLabels function A-76
UiIdentificationBoxCreate function A-77
UiInformationBoxCreate function A-78
UiIntegerFillinCreate function A-79
UiIsDefaultButton function A-80
UiLabelCreate function A-81
UiMenu
  Source(cust) 5-38
UiNonAligningFormCreate function A-82
UiObject
  ActiveWindow 5-3
  AssistLIne(cust) 5-5
  Cursor 5-13
  Dependency 5-17
  DestroyCb 5-18
  EditorIcon 5-19
  HelpContext 5-22
  Icon(cust) 5-23
  IconifiedIcon(cust) 5-23
  IconifiedTitle(cust) 5-23
  Name 5-30
  OtherData 5-31
  Sensitive(cust) 5-37
UiObjects
  AlignCmdBox 1-16
  AlignForm 1-16

ApplyPulldown 1-18
CommandBox 1-16
customizing
designing a GUI 1-2
DialogBox 1-16
EditorDialogBox 1-16
Fillin 1-17
FloatFillin 1-17
calculator 1-17
IntegerFillin 1-17
calculator 1-17
Label 1-18
NonAlignForm 1-16
OptionList 1-18
Paragraph 1-17
PushButton 1-18
RadioSet 1-17
ScrollList 1-18
ScrollMatrix 1-17
Separator 1-18
SlideOffEditorDialogBox 1-16
Slider 1-18
Toggle 1-18
UiObtainEntry function A-83
UiOptionListCreate function A-84
UiPaletteCreate function A-85
UiParagraphCreate function A-86
UiParagraphInsertText function A-87
UiParagraphMoveCursor function A-88
UiPostCab function A-89
UiPromptOn function A-90
UiPushButtonCreate function A-91
UiQueryItem function A-92
UiQueryResultBoxCreate function A-94
UiQuestionBoxCreate function A-95
UiRadioScrollListCreate function A-96
UiRadioSetCreate function A-97
UiReset function A-98
UiResetCab function A-99
UiScrollListCreate function A-100
UiScrollMatrixCreate function A-101
UiSeparatorCreate A-102
UiSet... function A-103
UiSet...Cb function A-105
UiSet...FromMessage function A-120
UiSetBorderWidthForSldoffDlgBox function A-104
UiSetCell function A-107
UiSetCellColor function A-108
UiSetCellValues function A-109
UiSetChosen function A-110
UiSetColumnAlignments function A-111
UiSetColumnColors function A-112
UiSetColumnLabel function A-113
UiSetColumnLabels function A-114
UiSetDefaultButtonWithSThickness A-115
UiSetDependency function A-116
UiSetEditorAttributes function A-117
UiSetEntryLabel function A-118
UiSetEntryRequired A-119
UiSetGeometrySpecified function A-121
UiSetInstanceId function A-122
UiSetItemSensitive function A-123
UisetMatrix function A-124
UiSetRowColor function A-126
UiSetRowColumn function A-127
UiSetRowLabel function A-128
UiSetRowLabels function A-129
UiSetSpecifiedPosition function A-132
UiSetSpecifiedSize function A-133
UiSlideOffDialogBoxCreate function A-130
UiSlideOffEditorDialogBoxCreate function A-131
UiSpecifierCreate function A-134
UiStartBusy function A-135
UiToggleCreate function A-136
UiToggleEntryCreate function A-137
UiWarningBoxCreate function A-138
UiWorkingBoxCreate function A-139
UserText(cust) attribute 5-41
IdentificationBox 5-41
text for the user to add through attribute customization 5-41
Using
menus in CADDS
preparation 4-3
sample menu 4-9
Using the mouse
adjust 1-14
menu 1-14
select 1-14
Utility
HLI String 3-22

V

Visibility
customizing 3-30
Visible attribute

Attribute
Visible 5-41
BuildingBlock 5-41
Sets the VisibleWhenManaged attribute 5-41
VisibleNow attribute 5-42
BuildingBlock 5-42
specifies whether building block is visible currently 5-42
VisibleWhenManaged(cust) attribute 5-42
BuildingBlock 5-42
CommandBox 5-42
DialogBox 5-42
displays the object 5-42

W

WarningBox class 6-57
displays warning messages 6-57
Width attribute 5-42
BuildingBlock 5-42
current width of the building block 5-42
WidthSpecified attribute 5-43
BuildingBlock 5-43
DialogBox 5-43
the application's preference for initial box width, exclusive of window decorations 5-43
WindowActivateCb callback 5-43
called when focus of attention moves from one dataset window to another 5-43
UiCallback 5-43
Windows NT
dynamic loading of custom menus 4-8
using custom menus with CADDS
dynamic loading 4-8
WindowTitle(cust) attribute 5-43
AlignCmdBox 5-43
CommandBox 5-43
DialogBox 5-43
displays the title at the top of the window 5-43
FromMessage convenience functions 5-43
TopLevel 5-43
WorkingBox class
shows task in progress 6-58

X

XPosition attribute 5-44
BuildingBlock 5-44
current horizontal position of the upper left corner of the building block 5-44
Xresource file 4-13
XSpecified attribute 5-44
BuildingBlock 5-44
DialogBox 5-44
specifies horizontal location 5-44

Y

YPosition attribute 5-44
BuildingBlock 5-44
current vertical position of the upper left corner of the building block 5-44
YSpecified attribute 5-45
BuildingBlock 5-45
DialogBox 5-45
specifies vertical location of application or dialog box 5-45